SESSION MANAGEMENT FOR ENABLING MULTI-PARTY INTERACTIVITY AND LARGE-SCALE VIEWING IN 3D TELE-IMMERSION

BY

MD AHSAN AREFIN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor Klara Nahrstedt, Chair
Assistant Professor Matthew Caesar
Associate Professor ChengXiang Zhai
Doctor Jin Li, Principal Researcher and Manager, Microsoft Research

Abstract

Over the last few years, we have seen a surge of interest in telepresence video collaborative technologies where remote users perform virtual-reality-like interactions with each other using real-time 3D video. Though the initial focus was on video conferencing applications, recent developments in audio-video and network technologies have expanded the horizon by supporting full-body interaction for many other physical activities in the virtual environment. Despite great potential, 3D tele-immersive (3DTI) sessions still face significant challenges due to high interactivity, multi-stream dependency, multi-view dynamism and large resource demand. Quality of service (QoS) allocation in 3DTI sessions not only depends on the availability of network and processing resources, but also on the users' expectations, which are dynamic and heterogenous.

In this dissertation, we revisit the design space of 3DTI session management framework. We propose a novel, comprehensive, user-experience-aware, programmable, cross-layer session control framework for 3DTI applications. Our control framework consists of a distributed monitoring oracle, a decision engine for computing user-experience-aware multi-stream dissemination topology, and a controller for allocating resources and configuring application and network layer data plane at the users.

The monitoring oracle allows run-time session queries by arranging application and system layer metadata over a tree-based overlay. It resolves multi-attribute range-based performance queries in real-time without interfering application data traffic. Based on the monitored resources, performance queries, and user demands, we compute a multi-stream content dissemination topology and allocate QoS among the users. Due to the differences in interactivity requirement and scale, we consider separate content distribution solutions for immersive users (who interact with each other in the shared virtual environment in real-time) and non-immersive users (who watch the collaborative activities performed by the immersive users, but do not immerse themselves into the virtual world). The number of immersive users are small, however, the number of non-immersive users can be in the order of thousands.

The immersive user prioritizes interactivity and drops delayed streams to ensure consistency across multiple geographically distributed streams. QoS allocation in this domain influences 3DTI experience of the users based on the on-going 3DTI activity. Different activities require different performance QoS profiles to ensure strong quality of experience (QoE) of the users. We model the problem of multi-stream QoS allocation as a prioritized multi-objective optimization problem, and solve it using an evolutionary approach. We show that the prioritized optimization of QoS meets expectations of the users up to 50% higher compared to the existing content distribution

solutions in the 3DTI space. To further improve the interactivity in the immersive session, we introduce application and network layer synergy. It reduces multi-stream management overhead of the application nodes by splitting the application data plane responsibility, and offloading it partially to the network layer. To control the network layer components during the session run-time, we leverage support from Software Defined Networking (SDN). We show that our solution improves 3DTI interactivity and resource usage at the data plane compared to the current solutions. Furthermore, it keeps data plane robust and ensures seamless visual experience against host failures and frequent topology changes during immersive sessions.

On the other hand, the non-immersive users prioritize the visual quality at the users rather than end-to-end latency. It requires to preserve multi-party multi-stream visual dependency at the display in addition to meet the usual challenges available in 2D VoD (video-on-demand) applications. Buffers are not enough as the number of streams and the size of 3D frames are both large. To solve the multi-stream content distribution and dependency preservation problem, we first design a CDN-P2P based hybrid multi-stream content distribution structure by considering users' views, stream priorities, network link bandwidth, and end-to-end delay of the streams. Second, we develop a distributed content caching technique, by organizing the non-immersive users in a logical hierarchy, based on multi-stream latencies. We show that our solution improves viewing quality and improves bandwidth utilization up to 55%, and supports a large number of concurrent non-immersive users.

To my parents, for their love and support.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my advisor, Professor Klara Nahrstedt, for initially giving me the opportunity to join this fascinating tele-immersion project, for pushing me forward while giving me the freedom to go for things that excited me, for all the valuable discussions and countless comments on my research, for the willing advice whenever needed, and for the continuing support, belief, and encouragement. I will be forever grateful for having Klara as my advisor.

Besides my advisor, I would like to thank the rest of my thesis committee members, Associate Professor ChengXiang Zhai, Assistant Professor Matthew Caesar, and Doctor Jin Li, for their encouragement, insightful comments, suggestions and hard questions. I would also like to thank Associate Professor Indranil Gupta for showing me how to perform good research during my early stage of the PhD study. I had privileges to attend courses with Associate Professor Indranil Gupta, Professor Jiawei Han, Professor Dan Roth, Professor David Nicol, Professor Josep Torrellas, Assistant Professor Sayan Mitra, Assistant Professor Matthew Caesar and Professor Klara Nahrstedt, which improved my intellectual ability.

I would like to thank to my colleagues in the tele-immersion project, particularly Raoul Rivas, Wanmin Wu, Zixia Huang, Pooja Agarwal, Shu Shi, Zhenyu Yang, Renata Sheppard, Pengye Xia, and Chien-nan Chen for offering generous assistance in my experiments, brainstorming with me on new ideas, and proof-reading my papers. I would also like to specially thank our collaborators at University of California at Berkeley, Gregorij Kurillo, Ramanarayan Vasudevan, and Professor Ruzena Bajcsy. I was able to run distributed experiments using their labs remotely.

I am thankful to the present and past members of Multimedia Operating Systems and Networking (MONET) research group including Hoang Nguyen, Ying Huang, Long Vu, Qiyan Wang, Thadpong Pongthawornkamol, Rahul Malik, Shameem Ahmed, Naveen Cherukuri, Hongyang Li, Haiming Jin, Zhenhuan Gao, Anjali Sridhar, Aadhar Jain, Andrew Kryczka, and Kurchi Subhra Hazra. They consistently attended my research talks and provided insightful comments. Sincere appreciation is also extended to Lynette Lubben, Andrea Whitesell and Mary Beth Kelley who helped me in administrative matters.

I am thankful to my NEC Labs mentor Doctor Geoff Jiang, who provided me an opportunity to work in his group over summer 2010 and 2011. His direction and advise added a new dimension in my research. He is also a real educator. I am also thankful to Doctor Paul England and Dan Behrendt from Microsoft Research. I was lucky to engage in an exciting research with them during summer 2012.

I am grateful for all the friends with whom I spent my time at UIUC, who made my life in this flat land fun and memorable. I specially like to thank Atanu Khan and Fariba Khan for providing me mental comfort as my local guardian. I am also grateful to Imranul Houe, Sonia Jahid, Maifi Hasan Khan, Liza Jahid, Shamsi Tamara Iqbal, Munawar Hafiz and Farhana Ashraf for extending their helpful hands starting from my first day in Champaign-Urbana and keeping them until the end. My friends have

amazing virtue of keeping me activite under stressful situation. I would like to mention my special thanks to Mehedi Bakht, Ahmed Khurshid, Yusuf Sarwar Uddin, Abdullah Mazahid, Wasim Akram, Muntasir Rahman, Abdullah Al-Nayeem, Mazhar Islam, Reaz Mohiuddin, Ashiqur Rahman, Kallol Das, Shama Farabi, Hasib Uddin, Shahala Chowdhury, Ibtesam Nazim Ahmed, Sadia Zabeen, Abdul Mannaf Rahat, Mahi Karim, Abul Hasan Samee, Shameem Ahmed, Moushumi Sharmin, and Gourab Kundo. Beside students, senior professional members of Bangladesh Community in Champaign-Urbana were always there for me. I am grateful to Professor Salim Rashid, Professor Taher Saif, Tarek bhai, Tuhin Bhai, Nadeem bhai, and Sharif bhai.

Above all I am thankful to the Almighty for blessing me with a wonderful family. I am grateful to my mom Halima Rahman and my dad Md Abdur Rahman for their continuous support, without which none of my achievements would have been possible. I feel deeply indebted for their understanding, unconditional support, and countless sacrifices to give me the best possible education. I therefore dedicate my thesis to them. I also thank my parents-in-law for continuously providing me mental support over the phone during my critical years of PhD. I am lucky to have two wonderful brothers and one loving sister. This thesis would not have been possible without their consistent support in many ways. I am grateful that my twin brother Md Minhaj Arefin agreed to study medicine, which motivated me to select engineering study. My elder brother Md Forhad Rabbi and loving Youngest sister Fayika Farhat were always been there for me during the good times and the bad times. They have the amazing ability to make me laugh even when I am depressed.

Finally, I would like to express my gratitude to my beautiful and caring wife Rehana Tabassum. She always kept me motivated during the journey of my PhD. She not only celebrated my success but also cherished me in my failure. She bear with me even there were many busy days I could not even talk to her properly. Truly, she has enlightened my life in many ways. Words can hardly express how grateful I am for having her in my life.

This material is based in part upon work supported by the National Science Foundation under Grant Numbers NSF CNS 05-20182, NSF CNS 07-20702, NSF CNS 08-34480, NetSE 1012194, NetTS 0964081 and CSR 0834480.

Table of Contents

List of Tables

List of Figures

LIST OF ABBREVIATIONS

GSC	Global session controller that represents a global coordination plane
LSC	Local session controller that manages local session at the users
$GrSC$	Group session controller that manages a set of LSCs
NC	Network controller that controls network switches at the users
T	Monitoring tree overlay constructed using distributed LSCs
$root(T)$	Root of monitoring overlay tree T
$p(X)$	Parent of node X in the monitoring overlay tree structure
$C(X)$	Set of children of node X in the monitoring overlay tree structure
k	Degree of nodes in the monitoring tree overlay
$r(l, h)$	Range assigned to the monitoring overlay from l to h
a	Monitoring attribute (or metadata)
v	Value of a monitoring attribute
d	Metadata item that contains a collection of (a,v) pairs
$\delta^a(X)$	Range assigned to node X for metadata a
$\Delta^a(X)$	Sub-tree range assigned for a tree rooted at X for metadata a
LD	Imbalance factor in terms of metadata storage load ratio
λ	Metadata update rate in a running session
μ	Metadata query rate in a running session
$q(a, r)$	Query for metadata items with attribute a having value within range r
S_i	i^{th} input stream
S_i^X	i^{th} input stream generated from Site-X
x_i	i-th controllable QoS parameter
r_i	Requested set of streams by Site-i
o_i	Generated set of streams from Site-i
$G(V, E)$	Graph with participant set V and network links E
v_i	Gateway of Site-i indicating a vertex in V

e_{ij}	Connecting network path between v_i and v_j
d_{ij}	End-to-end delay associated to e_{ij}
α	3DTI activity
min_x^α	Minimum quality of x for activity α
max_x^α	Maximum quality of x for activity α
p_x^α	Priority of x for activity α
Pr_s^A	Priority of stream S at Site-A
c_i	i-th chromosome representing a multi-stream distribution graph
ρ_i	Rank vector of chromosome c_i
Γ	Crowding distance of chromose c_i
R_s	Allocated rate for stream s
NVS	Number of video streams received per remote site
NSC	Number of stream channels received per remote site
NQ	Normalized cQoS value of multi-stream graph
u	Non-immersive users
C_{obw}^u	Outbound bandwidth capacity at non-immersive user u
C_{obw}^u	Inbound bandwidth capacity at non-immersive user u
C_{obw}^{cdn}	Outbound bandwidth capacity of CDN
obw_{cdn}	Outbound bandwidth allocated at CDN
obw_u	Outbound bandwidth allocated at non-immersive user u
ibw_u	Inbound bandwidth allocated at non-immersive user u
d_{buff}	Buffer length in non-immersive users
d_{skew}	Allowable skew between video streams
$N_{accepted}^u$	Number of accepted streams at non-immersive user u
N_{total}^u	Number of total streams rquested by non-immersive user u
AR	Acceptabce ratio = $\frac{N_{accepted}^u}{N_{total}^u}$
σ	Layer width in delay layer hierarchy
Ω	Minimum delay from immersive users to the non-immersive users via CDN
d_{max}	Maximum allowed end-to-end delay of streams
$d_{S_i}^u$	End-to-end delay of S_i from immersive users to non-immersive user u
bw_{S_i}	Bandwidth requirement of S_i
$obw_{S_i}^u$	Outbound bandwidth allocated for S_i at non-immersive user u

$oDeg^{u}_{S_i}$	Number of out-degree in the streaming overlay for S_i at user u
$Parent^{u}_{S_i}$	Parents of user u in the streaming tree for stream S_i
$\boldsymbol{Child^{u}_{S_i}}$	The child set of user u in the streaming tree for S_i
fr	Frame rate
$Layer^{u}_{S_i}$	Layer of stream S_i for non-immersive user u
d_{prop}	Stream propagation delay from one non-immersive user to another
ω	Delay induced by the software stack inside the gateway
$\Re$	Random offset generator
d_{cache}	Size of cache at the non-immersive user
κ	Layer bound
SRT	Session Routing Table
$gSRT$	Gateway Session Routing Table
FT	OpenFlow Flow Table
M	Adjacent matrix represnting a communication graph

1 Introduction

3D tele-immersive (3DTI) systems focus to provide an effective platform for multi-view immersive social interactions among geographically distributed users (or sites). They are much more media enriched than traditional audio-video systems such as Skype, PPLive [1], CoolStreaming [2], LiveSky [3] and YouTube. Examples of advanced TI systems are TEEVE [4] from UIUC, Halo from HP [5], and TelePresence from Cisco[6]. Though such technologies have been available in the market for several years, their overall session construction and management processes are still labor intensive and incomplete. They still rely on older session management protocols and standards such as IETF SIP [7], RTP/RTCP [8], and ITU H.323 [9], which do not efficiently take into account new requirements coming from 3DTI environments. In this dissertation, our goal is to design and develop a comprehensive session management framework for 3D tele-immersion. Before, we present our session management efforts, here we first give an overview of 3D tele-immersion and present the research challenges in the session management process. Finally, we provide an overview of our solutions and outline major contributions.

1.1 3D Tele-immersion

1.1.1 System Model

The 3D tele-immersive system is a distributed platform that connects multiple remote sites (or users) containing a large number of input and output devices (as opposed to the traditional video conferencing solutions), and creates a shared virtual space as shown in Figure 1.1. The system is usually composed of three architectural tiers: **capturing tier**, **stream dissemination tier**, and **rendering tier**.

- In capturing tier, multiple capturing devices such as 3D cameras (capturing separately the upper body and the lower body of the participants or users), microphones, and other haptic sensors capture the cyber-physical multi-modal information of each user at his/ her physical site. The captured streams are sent to a local *rendezvous point*, called site-gateway, which is responsible for data (stream) dissemination.
- The dissemination tier consists of an overlay network of gateways multiplexing streams to and from each site. A global controller is used to manage the gateways and to construct a global multi-stream dissemination topology among them. The gateway is responsible for both data dissemination and data reception.

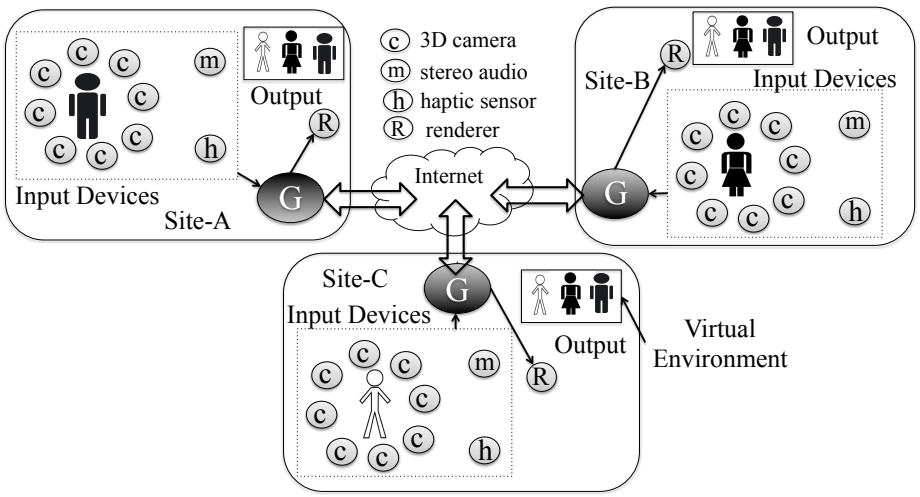


Figure 1.1: Architecture of a 3DTI system with three sites (participants or users).

- In the rendering tier, multiple rendering devices such as video displays, audio speakers and different haptic devices are used. The gateway sends both local and remote streams to the rendering tier to generate a combined perceptual feeds. All streams are rendered into a joint virtual world as depicted in Figure 1.1.

1.1.2 Data (stream) Model

Unlike conventional multi-view video conferencing systems, each camera in 3DTI is a stereo unit, typically equipped with binocular or trinocular lenses, and connected to a host computer via IEEE 1394 (FireWire) interface. The host computer grabs image frames synchronously from all lenses and produces a stream of *color-plus-depth* frames. A hardware trigger is used to synchronize all local cameras to grab images at the same instants of time. Individual video streams generated from a site are not merged at the source site, rather disseminated separately via the local gateway, because merging the depth maps of multiple streams to create a redundancy-free 3D model is still an unsolved challenge, particularly given the stringent latency requirement of interactive applications. Moreover, the size of the merged 3D model becomes very large, which invalidates the notion of dropping less important streams in case of network congestion. Hence, streams are merged at the rendering tier of the receiving sites. We use UDP protocol to transmit media streams, however the control traffic for the management and control running session uses TCP.

Not all streams generated by the input devices are required by all remote users during a 3DTI session. The demand for video streams depends on the view orientation of the users. For example, if a viewer is currently looking at the front of a 3D object, video streams generated by the back cameras are less important and can be dropped. The 3DTI also introduces a new notion of user interactions, where users can frequently change their view orientations in the virtual space. The change in the user views changes importance of the particular streams.

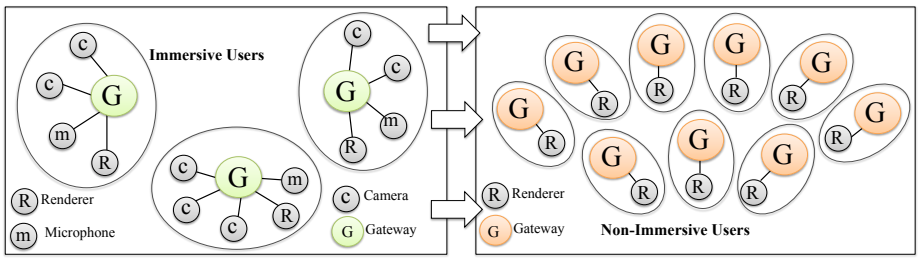


Figure 1.2: 3DTI content generation by immersive users and content viewing by non-immersive users.

1.1.3 User Model

There are two types of users in 3DTI session: *immersive users*, and *non-immersive users*. Immersive users are the active users who perform collaborative activities with each other in the shared virtual environment, e.g., dancers in collaborative dancing and players in immersive gaming. On the other hand, non-immersive users are passive users who view the joint performance of the immersive users without contributing in the virtual world with 3D contents. Examples of non-immersive users are audiences of virtual collaborative dancing or viewers of online exer-gaming. Real-time interactivity is the most prominent requirement for the immersive users. The number of immersive users in a 3DTI session is usually limited for two reasons: 1) human attention space is intrinsically bounded [10], and 2) the number of participants that can be displayed in the virtual space is limited due to the pixel space limitation of the physical display [11].

Unlike the traditional TV or IPTV viewers, the non-immersive viewers receive multiple streams at the same time from different geographically located sources (i.e., immersive users), and the viewers are able to change views of the 3D video. On the other hand, TV and IPTV viewers are subscribed to only one stream at a time and they rely on the providers (e.g., ESPN or FOX) to change views. Visual quality is considered as the most prominent requirement for the non-immersive users. The number of non-immersive users can be very large (in the order of thousands). Figure 1.2 shows the interaction between the immersive users and non-immersive users.

1.2 Research Problem and Challenges

Both industry and academia have adopted multi-camera, multi-site, and multi-modal 3D tele-immersive platforms for various Internet applications such as multi-player gaming [12], collaborative dancing [13], cyber-archeology [14], and rehabilitation [15] in addition to video conferencing [16]. However, the session management of these advance 3DTI applications is still problematic and challenging because of the following characteristics present in 3D tele-immersions.

(a) (b) (c)

Figure 1.3: Examples of multi-site 3D immersive activity: (a) video conferencing, (b) collaborative dancing, and (c) virtual lightsaber dual.

1.2.1 Heterogenous User Expectation

Even with wider applications, most of the 3DTI systems available on the market are optimized for one particular activity to achieve an optimal performance for the intended activity (e.g., Xbox with Kinect is optimized for gaming). However, in the recent years, we have seen that the same 3DTI platform can be used for multiple activities. Examples of diverse activities on the same 3DTI platform are shown in Figure 1.3, where geographically-distributed participants are engaged in video conversation, collaborative dancing and virtual multi-player gaming using the same 3DTI setup [17].

Different activities define different minimum quality bounds on quality of service (QoS) values in network and application levels to meet expectations of the immersive users [18], which manifest themselves in the form of quality of experience (QoE) [19]. For example, a TI conversation activity (where geographically distributed immersive users perform video conversation in a virtual room) requires a high quality audio stream, whereas the minimum quality requirement of the audio stream is low for a virtual lightsaber gaming [20] (where geographically distributed immersive users virtually fight with each other). On the other hand, the lightsaber gaming requires a high quality video stream, and very low end-to-end delay to allow smooth and fast remote interactions. In addition to the bounds in the minimum values, also the priority (i.e., importance) of the QoS parameters varies across the activities. For example, audio rate is more important QoS parameter than video rate in the video conferencing, whereas video rate is more important in the virtual lightsaber gaming. Such prioritized dependencies usually exist across all QoS parameters in running 3DTI sessions.

In summary, the streaming content, the minimum quality requirements for network and application QoS parameters and their priorities are highly dependent on the engaged 3DTI activities of immersive users. We call them **session parameters**. The prioritized optimization of QoS parameters given constraints on the session parameters is a NP-complete problem. Previous solutions [4][17] in the 3DTI domain focus on optimizing only one QoS parameter (the number of video streams). Moreover, the dependency among different QoS parameters is missing.

1.2.2 Frequent View Change

The state-of-the art stream selection approaches [4, 17] in the field of tele-immersion uses view-based stream priority for both immersive and non-immersive users. Fig-

ure 1.4(a) shows an example of multi-view 3DTI application. When network resources are limited, the lower priority streams are dropped to maintain high visual quality of the users. With the frequent view changes during the 3DTI session run-time, the priority of the streams requires frequent updates, which lead to the frequent modifications of the multi-stream content dissemination topology. In this dissertation, our goal is to develop a session management protocol to minimize the overhead and response time of frequent session updates and at the same time ensure optimized allocation of QoS parameters during the view change process triggered by the immersive and non-immersive users.

Modification of the multi-stream dissemination overlay introduces another challenge in the multicast-based dissemination topology. Let us consider an example with three sites (A, B, and C). If Site-C receives a stream of Site-A via Site-B in the overlay dissemination topology, the discard of the stream at Site-B due to a view change creates a disconnectivity of the stream to Site-C. Likewise, many other sites can become victims due to a view change. The session management protocol should ensure that the modification of the overlay due to the view changes (which define the stream priorities as one of the session parameters to consider in the session management process) do not create any **victim sites**.

1.2.3 Multi-stream Dependency

Streams generated from the same site are highly correlated, and they must be synchronized to ensure a consistent view of the virtual environment. This is a prominent requirement for both immersive and non-immersive users. However, the requirement of dependency preservation among the streams are more stringent for the non-immersive users. During the immersive session (session for the immersive users), no buffers are used to ensure high interactivity with low latency. Therefore, skews among the remotely correlated streams are kept bounded by reducing the end-to-end delay of the streams. The delayed streams are dropped to overcome the inconsistency.

On the other hand, the non-immersive session (session for the non-immersive users) puts high importance in maintaining remote stream dependency in addition to the local stream dependency without dropping the delayed streams to ensure good visual quality. Most of the large-scale video on demand solutions (specially with audio and video) solutions [2][3] use buffers. However, buffers are not enough here as in 3DTI, we require synchronization among multiple video streams in addition to the multi-modal streams, where the number of video streams and the size of 3D frames are both large. Therefore, our session management framework should construct a dissemination architecture that can assist in dependency preservation of the local and remote streams at the non-immersive users.

1.2.4 Large Resource Demand

Spatially, the use of 3D representation and multi-view capturing leads to a high demand on network bandwidth, because within a stream not only color but also depth information is encoded and multiple streams need to be sent from each site for different views.

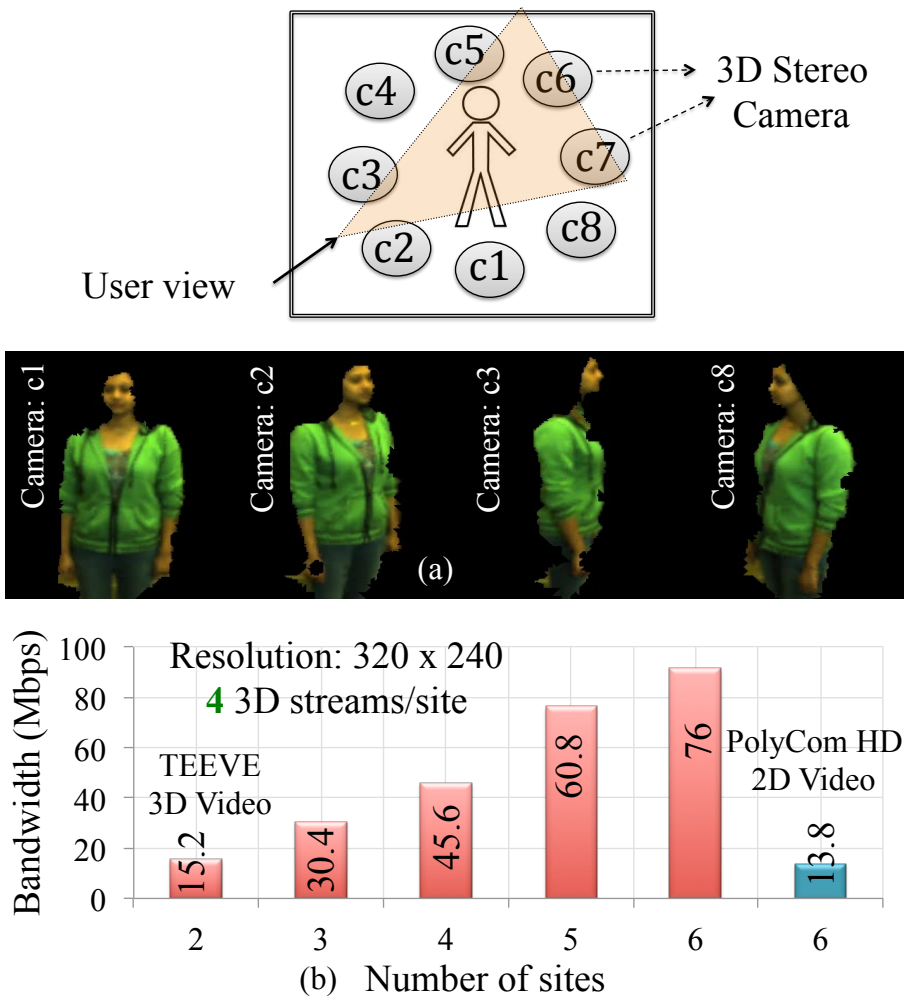


Figure 1.4: (a) Multi-view example in 3DTI. For the given user view, the priority of streams generated from camera $c2$ and camera $c3$ are more important. (b) Bandwidth requirement for TEEVE 3D video, and PolyCom HD 2D conferencing solution.

Using mesh-based reconstruction of image and 320×240 resolution, the size of each 3D frame can be up to 10Mbps depending on the quality of the reconstruction. Considering 10 3D cameras connected to a site, the total outbound for a single transmission of each stream requires up to 100Mbps bandwidth. Obviously, the aggregate resource demand directly depends on the amount of video data to process and/or disseminate.

In Figure 1.4(b) we show the bandwidth requirement in PolyCom HD 2D conferencing and TEEVE (Tele-Immersive Environment for EVErybody) [4] 3D tele-immersive applications for different number of users or sites. In this example, the optimized 3D reconstruction in TEEVE reduces the bandwidth usage to 3.8Mbps per stream [19], which leads to 76Mbps requirement with 6 sites, each containing 4 3D camera streams. On the other hand, the video bandwidth in PolyCom HD conferencing solution with 6 users always stays below 13.8Mbps. Therefore, network resources are very demanding in 3DTI applications specially in the *last mile* of the communication. In this disserta-

tion, we try to answer 1) can we allocate bandwidth efficiently to support large number of concurrent non-immersive users? and 2) is it possible to reduce network bandwidth usage at least at the edges (inside the last mile) during the immersive session by configuring the network layer?

1.3 Thesis Statement

In this dissertation, we revisit the design space of the session management process of 3DTI applications since current standards do not accommodate all requirements and challenges that 3DTI systems introduce. There are some discrete research efforts that try to address specific dimensions of the session management process, such as maximizing number of video streams [4], or supporting multiple immersive users [17], however, none of them are complete in terms of incorporating user heterogeneity and QoS requirements in a running session. Current session management protocols cannot handle heterogenous and dynamic user behavior, multi-view functionality, cross-layer control for session programmability, and large-scale correlated multi-stream dissemination. In this dissertation, we introduce heterogenous user behavior and dynamic expectations in the QoS allocation. Though earlier works [21][19] build a comprehensive model for mapping QoS to QoE, the dependency on the users' activity is missing. We look into the network layer to construct a synergy with the session layer in a feasible way to improve interactive latency and resource consumption on top of what the session layer can optimize. We show that we can enable a synergy between session and network layers so that application can drive network layer to support different application features. Overall, our dissertation goal of the session management process is to improve interactive end-to-end latency, resource allocation and seamless view change experience in both immersive and non-immersive sessions, and large-scale support with emphasis on preserving the temporal dependency among maximum number of 3D video streams in non-immersive session.

Thus our thesis statement becomes

> *3D tele-immersion must consider user-experience-aware, programmable, cross-layer session control framework to support user heterogeneity, QoS dynamism, frequent view changes, multi-stream dependency and high resource utilization.*

1.4 Our Approach

1.4.1 Session Management Functionality

This thesis essentially proposes a comprehensive, session management framework for managing multi-stream and multi-site 3D tele-immersive applications that prioritizes interactivity among immersive users and large-scale viewing for non-immersive users. Our approach is comprehensive because it involves all components of the session man-

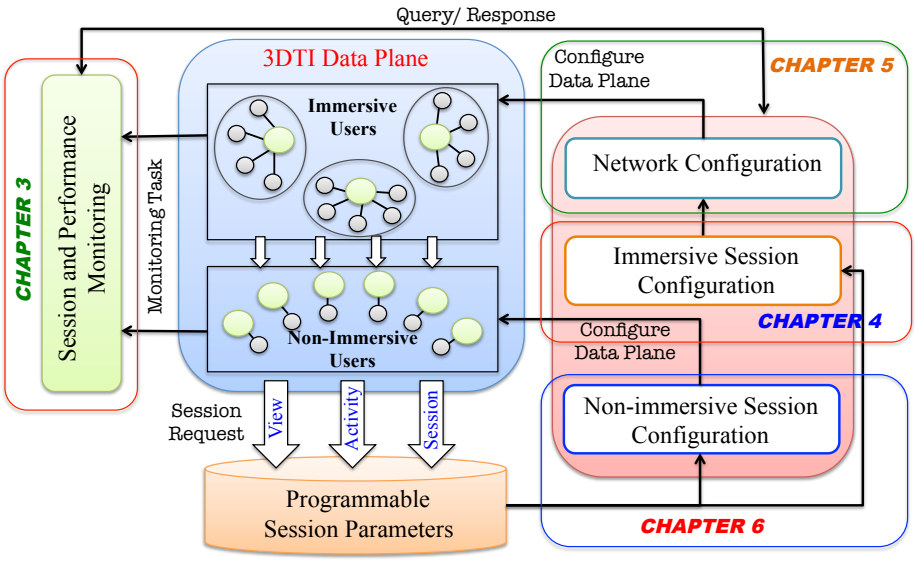


Figure 1.5: A Comprehensive Framework for 3DTI Session Management.

agement pipeline: session monitoring, QoS allocation and topology construction, and data plane configuration for QoS enforcement for both immersive and non-immersive users. Figure 1.5 gives an overview of the session management control framework. Below we present different functional components of our solution.

Session Monitoring: Session monitoring is an integral part of any session management architecture, which assists in decision making by providing useful information about the running session. In addition to traditional single-attribute-based performance queries, a range-based multi-attribute query is frequently used to understand QoE of the users and QoS of the participating infrastructure. In order to serve queries efficiently, we develop **Q-Tree** [22] that builds a topology-aware tree overlay (a single overlay for multiple attributes) by connecting participating 3DTI sites (both immersive and non-immersive sites), and assigns dynamic range intervals to each overlay node (i.e., site) in a hierarchical manner. Monitoring metadata (e.g., bandwidth, latency, frame rate, etc.) is distributed based on this range structure, which is self-adaptable based on the load condition of the sites. Finally, Q-Tree uses optimized algorithms for query dissemination to ensure most up-to-date results in fast and efficient way based on the update rate of metadata in a given query, and the query rate. Q-Tree is robust against node failure and self-configurable at session run-time. In Chapter 3, we present the design and performance of Q-Tree.

Immersive Session Configuration: To configure an immersive session, we consider activity dependent QoS priority and QoS bounds (known as session parameters as shown in Section 5.3.2). The session management goal becomes constructing a multi-stream topology with optimal QoS allocation. Our solution is called **Optimized Immersive Session Management (OSM)** [23]. We model the QoS optimization problem given QoS specifications (session parameters), resource constraints and delay bound

as a multi-objective optimization problem and solve it using a genetic algorithm (GA). GA uses an iterative approach and therefore, may include a considerable latency in the search of an optimal solution (i.e., a multi-stream content distribution topology with optimal allocation of QoS). However, using a careful selection of initial solution (i.e., multi-stream multi-site overlay graph satisfying minimum QoS quality and resource constraints) in the search space, and allowing topology diversities in the optimality search, we show that our solution converges fast towards optimized QoS allocation. Furthermore, to mask the optimization latency while updating a session during runtime, we develop a two-step management process. In the first step, an instantaneous non-optimal solution is installed to meet the demand of the session update while the optimal solution is computed in background. The second step installs the optimal solution. Details are given in Chapter 4.

Network Configuration: Configuring the overlay and allocating QoS according to the activity-based specifications are not the boundary for improvement in streaming applications. We can achieve further improvement in the immersive session by configuring the network layer. In Chapter 5, we therefore, shift our focus to the network layer. With the advancement of Software Defined Networking (SDN) research over the past few years, network components have become accessible and controllable from application layer [24]. Applications can use this flexibility to offload certain data plane functionalities to the network layer for the purpose of improving data plane complexity and efficiency. The natural question for us is "can we use OpenFlow network components to reduce data plane complexities and improve multi-stream flow management in 3DTI?". To answer this question, we develop **OpenSession** [25], a 3DTI network configuration protocol. In this dissertation, we propose 3DTI network configuration protocol only for the immersive session leaving the network configuration of the non-immersive session for future works. OpenSession introduces a split forwarding architecture at the application data plane. It *partially* offloads stream multicast functionality of the application to the SDN switches. If multiple participants request the same streams, the source participant sends only single copies of the local streams to the SDN network switch, which handles multi-stream forwarding to all remote destinations on-behalf-of the application. The splitting of data plane reduces processing load at the application, and network bandwidth usage at the network. Offloading multi-stream forwarding to the SDN network component also improves end-to-end delay in multicast-based streaming (only in immersive session) because forwarding of streams is done from the network layer of the SDN switches instead of from the application layer of the end-hosts at each multicast hop.

Non-immersive Session Configuration: The last part of the session management process provides support for the non-immersive users. During non-immersive sessions, it is possible to relax end-to-end delays (though bounded) while distributing multi-streaming contents in order to improve visual quality, handle churns and accommodate large scales. To support a large-scale, we consider a hybrid (CDN-P2P) dissemination structure with P2P overlay constructed providing incentives: the participant with higher

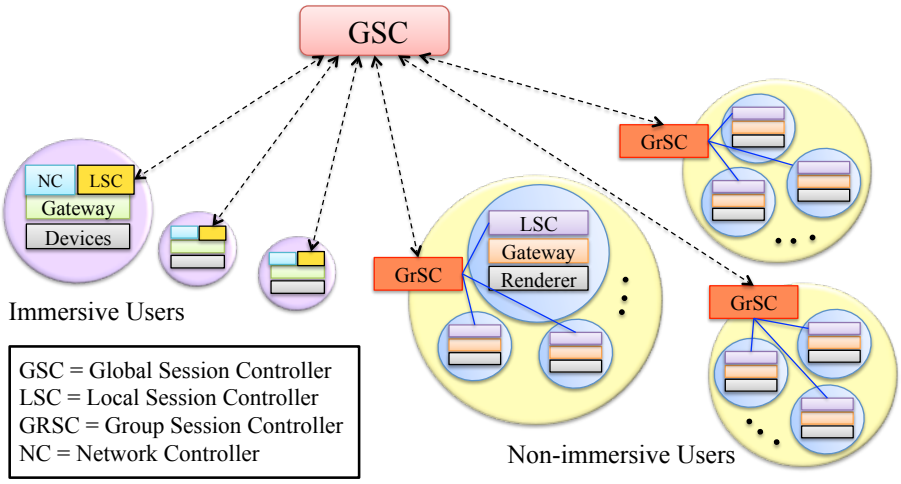


Figure 1.6: Session Management Architecture.

available outbound bandwidth is set higher in the distribution tree (i.e., lower end-to-end latency). To preserve the multi-stream dependency (shown in Section 5.3.2), we organize the non-immersive users in a hierarchy of logical layers, for each stream they receive, depending on the end-to-end latency and its variations perceived at the users. Based on this layer information, streams are cached at parents so that streams can be delayed in the forwarding to ensure a correlated consistent display. Our solution is called **4D TeleCast** [26] since we focus on tele-immersive telecast of 3D contents, where the contents are defined based on the view orientation (which is the fourth dimension) of the users. Details of large-scale content dissemination and stream dependency preservation for the non-immersive users are given in Chapter 6.

1.4.2 Session Management Architecture

To perform the session management functionalities shown above, we construct a hierarchical session management architecture with several components as shown in Figure 1.6. A subset of the components are used to achieve different functionalities.

- **Global Session Controller (GSC)** is at the top of the management hierarchy. It is a global coordination point for all session management components. GSC is responsible for understanding the session requirement, finding spatial relationship among the streams and finally define the programmable session parameters. For the immersive session, GSC also computes multi-stream overlay dissemination graph with optimal QoS allocation.

- **Group Session Controller (GrSC)** is used only for the non-immersive session to scale the management functionality for large-scale of non-immersive users. Each GrSC maintains a cluster or group of non-immersive users and is responsible for computing multi-stream overlay dissemination graph with optimal resource allocation only over the group.

- **Local Session Controller (LSC)** configures the 3DTI data plane. It is also responsible for monitoring the data plane and the underlying infrastructure. LSCs maintains *session routing tables (SRT)* at each site, which gateways use to route streams. The fields of SRT vary across the immersive and non-immersive sessions. LSCs are usually reside within the gateway host.

- **Network Controller (NC)** is used to configure the network components. In this dissertation, we only consider network configuration at the immersive users. The goal of network configuration is to enable IP layer multicasting at the network edges in a feasible way. It is also responsible to ensure seamless streaming experience during session modification, which requires modification in the distributed network components inside each participating sites.

1.5 Major Contributions

Our major contribution is designing a comprehensive session management framework for both immersive and non-immersive 3DTI sessions that incorporates dynamic user expectations in resource allocation, yet ensures efficient resource utilization, high interactivity and seamless experience in session adaptation. Other contributions are below:

- Queries in 3DTI are not like traditional database queries with a single key value (hence, no DHT structure [27][28] will work), instead they are given in a high level description, which are transformed into multi-attribute composite range queries. To the best of our knowledge, Q-Tree is among the first that considers a single overlay to monitor multiple attributes (metadata) in a hierarchical fashion and at the same time ensure minimum traffic overhead and low service latency in answering multi-attribute range (e.g., MAX, MIN, AVG, SUM) queries without interruption application traffic (Section 3.3 - Section 3.4).

- We are also among the first to study the impact of 3DTI activity on the mapping of QoS to QoE. We show a novel usage of genetic algorithm in the session management process to optimize QoS considering the activity-defined requirements to maximize heterogenous user expectations (Section 4.5). Our solution meets expectations of the users up to 50% higher compared to the existing content distribution solutions in the 3DTI space (Section 4.8).

- Though different IP multicasting protocols (e.g., PIM-SM [29], MBONE [30]) are available for streaming, they are not in practice due to the lack of control from the application layer and feasibility of deployment. In our dissertation research, we successfully enable IP multicasting over some parts (last miles) of the Internet end-to-end data transmission path using Software Defined Networking (SDN). We show a synergy between session and network layers so that application can drive network layer to support different application features. We are the first in our knowledge to show the feasible usage of SDN for real-time traffic management in the interactive collaborative application domain (Chapter 5). We

believe that our results will lead many future research efforts to control QoS and other application features in the network layer using SDN support.

- We propose cross-layer session-network control algorithm in the immersive domain to improve 3DTI latency, reduce bandwidth usage within the edge network and to minimize processing load inside the application host (Section 5.5 - Section 5.8). It makes the 3DTI data plane resilient against frequent view changes, route updates and host failures, and enables seamless session adaptation without creating any victim sites in the immersive domain.

- Though many great efforts have been performed to solve content distribution in the immersive domain, we are first to introduce multi-stream multi-view content distribution in the non-immersive domain. Our CDN-P2P based topology construction and bandwidth allocation do not require any administrative control inside the CDN, yet supports thousands of concurrent viewers (Section 6.3).

- We consider user expectations while designing caches for 3D frames during the non-immersive session. Our *delayed-streaming* solution in the non-immersive domain improves users' visual experience by preserving multi-stream dependencies (Section 6.4). Resources are de-allocated if the dependency cannot be preserved, which improves effective resource utilization over 55% depending on the allowed buffer size at the users (Section 6.5).

2 Literature Review

SIP (Session Initiation Protocol) is a session layer signaling protocol [7] used for session management in IP telephony and VoIP services. SIP does not provide mechanisms to configure data plane for streaming. Other session protocols like RTP (Real-time Transmission Protocol) [8] uses in-band control functionalities over the data channel to assist streaming, RTCP (RTP Control Protocol) [8] is out-of-band control protocol for RTP, and SDP (Session Description Protocol) [31] is used for exchanging session information among the users. RTSP (Real-time Streaming Protocol), the Real Time Streaming Protocol, is a client-server multimedia presentation protocol to enable controlled delivery of streamed multimedia data over IP network [32]. They all are point-to-point protocols providing end-to-end signals to perform session layer adaptation. H.323 is another session initiation and management protocol which is an entire suite of protocols providing functionalities ranging from call control, conferencing, and codecs to many other features [9]. The advantage of using H.323 is that it provides large amount of functionality in itself and hence, decreases the dependence on protocols like RTP, SDP, and RTSP. This in fact increases the overall performance of the protocol. However, in the world of ever changing IP telephony requirements, H.323 becomes lesser flexible and difficult to modify for emerging multi-stream multi-modal systems. None of the above standards can accommodate advanced requirements coming from the 3DTI applications. For example, the above protocols does not provide any provision for incorporating multi-stream priority and spatial relationships among them. Moreover, the heterogenous user expectations and dynamic view modification put unique requirements in the multimedia session management process that older protocols do not address. Below we provide more detail reviews of other related works by dividing them into different processes in the session management pipeline.

2.1 Session Monitoring

The problem of designing a distributed query plane comprises of two aspects: a monitoring overlay containing all participating sites and a distributed search engine to answer session related queries. Here, we review several systems to seek how they are related to the multiple metadata (or multiple attribute) based range queries in 3D teleimmersion. CAN [33], Chord [27], Kelips [28], Pastry [34] and Harren et al. [35] implement distributed hash structure to provide efficient lookup of a given key value. Since a hash function destroys the ordering in the key value space, these structures can not process range queries. P-Ring [36] supports both equality and range queries

Case	*Chord*	*P-Tree*	*P-Ring*	*Skip Graph*	*Q-Tree*
Overlay Choice	top-down	top-down	top-down	top-down	bottom-up
Performance	O($\log N$)	O($\log_d N$)	O($\log_d N$)	O($\log N$)	O($\log_{k-1} N$)
Value per-node	multiple	single	multiple	single	multiple
Range Query (R)	no	yes	yes	yes	yes
Multi-Attribute R	PAO	PAO	PAO	PAO	yes

Table 2.1: Comparative analysis of Q-Tree

with logarithmic search performance using hierarchical hash. But it does not support aggregation, and also it is designed for only single attribute range queries.

There have been some other systems that are built to generate aggregated results on top of a large distributed overlay. Astrolabe [37] provides the abstraction of a single logical aggregation tree with administrative hierarchy for autonomy and isolation. Scalable distributed information management system (SDIMS) [38] is implemented based on this approach and provides an aggregation abstraction on top of an overlay satisfying scalability, flexibility, autonomy, isolation, and robustness. SDIMS organizes nodes as an aggregation tree where leaf nodes are the physical machines of the system, and internal nodes, represented as virtual nodes, correspond to an administrative domain for information management. Astrolabe and SDIMS do not provide enough provision for efficient range queries. They were not developed to serve the purpose required by the 3D tele-immersive environments.

There are also other monitoring and management tools developed for PlanetLab [39]. CoMon [40] provides monitoring result both at a node level and a slice level. It is actually used to monitor the performance of nodes and to examine the resource profiles of individual experiments. Plush [41] is another application management tool for PlanetLab. It is designed to deploy and manage naturally distributed tasks. None of these suport aggregation or range queries. MON [42] is an on-demand monitoring service for PlanetLab. To reduce the cost of maintenance, MON constructs a multicast tree on the fly to serve a query. MON is a better solution for multicast rather than range queries, because it has no prior knowledge about the attributes. Moara [43] is able to resolve queries efficiently by lowering response time and reducing message overhead. But its composite query plane needs to be transformed into canonical form to provide lower latency which eventually increases the overhead. On the other hand, Q-Tree allows any form of boolean expressions to fuse as a query. Only a single query message is fused in the overlay even though the composite query may search for multiple metadata.

We analytically compare our system with P-Ring, P-Tree, Chord and Skip Graph. Although they have the same $O(\log N)$ performance, none of them originally supports multi-attribute range queries and in-network aggregation suitable for TI structure. Further Q-Tree uses single overlay structure for all attributes as opposed to the *per attribute overlay* (PAO) found in other approaches mentioned above. As, the overlay maintenance cost is amortized over all attributes, our approach provides scalability in terms of number of attributes. In Table 2.1, we give a comparative study among them

according to their basic models.

2.2 Immersive Session Management

QoS Allocation: QoS allocation in multi-stream topology construction considering user expectations is not new in the immersive 3DTI domain. ViewCast [4], for example, constructs a multi-stream, multi-site content distribution graph by maximizing the number of video streams per participant in order to improve 3D video quality. They consider stream priority (defined by the participant's view orientation) while dropping stream requests given bandwidth and delay constraints. Wu et al. [17], on the other hand, constructs a randomized content distribution graph. They consider node *criticality* while dropping streams, where the criticality for each remote user is computed as the reciprocal of the number of streams a participant requests from the respective participant. When a conflict occurs due to the resource constraints, the streams from the lower critical participants are dropped. They do not guarantee the maximization of the higher priority streams, rather they maximize the average number of streams from unique participants. Both [4] and [17] construct topology to maximize number of video streams. However, maximizing the number of video streams is not the only QoS factor that improves user experience. Other QoS parameters such as number of multi-modal channels, media rate, end-to-end latency, and synchronization skew highly influence the 3DTI experience [26]. Sometimes dropping of a video stream to optimize the rate of another stream can gracefully improve user experience.

QoS Optimization: There are also other works that consider constraint-based QoS maximization in network routing. Celerity [44] for example, constructs a multi-party multi-rate 2D video conferencing solution subject to bandwidth and end-to-end delay constraints. They formulate the problem as a rate maximization problem and provide a polynomial-time tree packing algorithm on the source and an adaptive rate control along each overlay link. However, they do not prioritize streams that share common resources and therefore, it is not applicable in the 3DTI domain. Some related literature can also be found on QoS routing for IP multicast [45] and overlay construction for application-level multicast (e.g., [46], [47], and [48]). However, none of them consider the complexity in constructing multi-stream dissemination structures, where the participating sites and resources are shared. Moreover, the priority of the QoS parameters in the 3DTI space varies depending on the activity a participant performs virtually.

2.3 Non-immersive Session Management

Large-scale Content Dissemination: Multicast routing algorithms have been well studied in [45]. [45] and [49] also point out that constructing a tree that optimize multiple metrics is an NP-complete problem. However, conventional tree based multicast may lead to a lower acceptance ratio [17][50]. [51] considers path delays of the requested streams for constructing the 3DTI dissemination overlay. The solution works

well for preserving the interactivity among the immersive users, but does not create optimal overlay for supporting large number of non-immersive viewers. P2P-based IPTV service delivery is a well-explored area nowadays and a number of solutions are available for large-scale delivary. CoolStreaming [2] is a framework known for P2PTV (peer- to-peer television) technology enabling users to share their television content with each other over the Internet. It is similar to the Bittorrent protocol [52], where participating peers upload their content over an overlay networks where each node periodically exchanges data availability information with other nodes for the video streaming. PPLive [1] is the most popular mesh based framework for P2P-based IPTV services. It implements a P2P based video distribution protocol and the management of peers and channel discovery is supported by a gossip based protocol. A central server keeps track of all the available chanles so, a new peer sends a query to central channel server to get the list of available channels while joining the network. There were also efforts based on the unstructed overlay [53][54]. However, the current IPTV solutions are missing two unique requirements in 3DTI: 1) the delivary structure should assist in preserving multi-video dependency, where the video streams are coming from different geographical regions, and 2) the video content should be organized to support multiple concurrent views at different users.

QoS Allocation: There have been some efforts, both in academia and industry, to address the scalability issue presented in the streaming media service using P2P networking techniques [55][56][57]. Data buffering at clients corresponds to interval caching that was first proposed in [58][59] to efficiently utilize memory for video streaming. It has also been applied at the application level in several occasions [56][57]. The goal was mainly to increase the data availability while supporting a large-number of clients. Rather in our session management process, we introduce caching to improve multi-video synchronization in addition to the buffer at the users. Unlike other approaches, we consider multi-stream caching. Therefore, the challenges to design cooperative caching are different, which are not addressed by the earlier approaches.

2.4 Network Layer Stream Management

IP Multicast Routing: IP multicasting protocols (e.g., PIM-SM [29], MBONE [30]) do not provide any application layer programmable interface at the routers and global control for run-time dynamic configurations. To physically deploy PIM-SM, a dependency across ISPs is required, which ISPs do not allow. MBONE overcomes the ISP dependency by creating a virtual multicast network. However, it requires a fixed additional backbone and does not scale. On the other hand, PIM-SM does not scale due to the flooding of bootstrap control messages. Via OpenSession, we successfully enable quality of service control over some parts (last miles) of the Internet end-to-end data transmission paths.

Network Flow Management: The efforts of cross-layer multimedia flow management can be divided into three categories: resource reservation (e.g., [60], [61]), rout-

ing (e.g., [62]) and flow scheduling (e.g., [63]). Unlike RSVP, OpenSession does not require control of network routers inside the ISPs for the feasibility concern. OpenSession considers layer-3 control over wide-area-networking and so, MPLS [64] and other layer-2 quality control protocols do not work. Overlay becomes our only choice for application routing. Along with different overlay routing approaches [65], various application layer resource reservation and scheduling techniques can still be performed within OpenSession framework.

Application Support with SDN: Software Defined Networking (SDN) has enabled network layer control the application layer. In SDN, switches are controlled and modified by a central controller, enabling flexible functions such as flow forwarding, redirection, multicast, routing, and others (e.g., [66], [67], [68]). SDN switches and their control have been used in data centers (e.g., [69], [70], [71], [72]) and assisting in scalable video streaming (e.g., [73]) via server redirection. However, our application domain is unique in several ways: 1) we require network control over wide area network, and 2) a consistent update across the distributed switches are required over the Internet. There has been much work on retaining consistency in network updates. oFIB guarantees an ordering of updates over a network to mitigate convergence-related outages [74]. Another work [75] improves consistency of SDN networks, at the cost of increasing state in switches by storing duplicate table entries. It does not support bandwidth or other metric guarantees during convergence. [72] aims to compute a sequence of migration so that network and hosts are not overloaded. However, the focus of our work is on consistent update considering network burst and temporal outage.

3 Session Monitoring

3.1 Introduction

Monitoring is an integral part of our session control framework. Both the immersive and non-immersive 3D tele-immersive sessions use the monitoring control plane for querying session performance. However, queries in 3DTI are not like the traditional database queries with a single key value, instead they are given in a high level description, which are transformed into multi-attribute composite range queries. Some of the examples include "which site is highly congested?", "which devices are not working properly?" etc. To answer the first one, the query is transformed into a multi-attribute composite range query with constrains (range of values) on CPU utilization, memory overhead, stream rate, bandwidth utilization, delay and packet loss rate. The later one can be answered by constructing a multi-attribute range query with constraints on static and dynamic characteristics of the devices. Queries can also be made by defining different multi-attribute ranges explicitly, e.g., "what is the list of cameras having frame rate less than 10 frames per second?". Due to the real-time nature of the system, the TI query plane should be lightweight in consuming network resources, scalable in terms of metadata items, and capable of answering queries in low latency. Here, we use the attribute and metadata interchangeably.

We propose Q-Tree [22], a multi-attribute range based query solution considering: 1) hierarchical nature of the 3DTI, 2) real-time requirement of the application data traffic, and 3) the need for multi-streaming range query. Q-Tree resides inside the distributed local session controllers (LSCs) (Figure 1.5) and constructs a monitoring overlay among them. The significant property of Q-Tree is that it injects only a single query to the monitoring overlay for any size of composite multi-attribute queries without any preprocessing and still ensures the optimal number of node (or site) traversal to answer the queries. It can handle significant amount of attribute churn in the TI system and also scales with the number of metadata items.

3.2 Monitoring Overview and Architecture

3.2.1 Overview

Q-Tree intents to provide a multi-attribute based query solution for hierarchically clustered environments. It organizes LSCs of the immersive and non-immersive users in a monitoring overlay tree structure and assigns ranges to them in some hierarchy. The

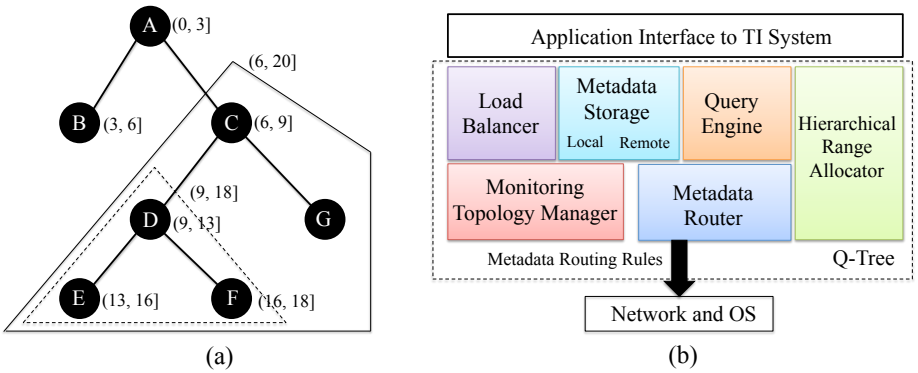


Figure 3.1: (a) Example of Q-Tree functionality. Each node in the monitoring overlay represents LSC of the participating users. (b) Q-Tree system architecture.

underlying requirement of serving queries is to retrieve metadata items from the participating sites. To assist it, the LSC of each site monitors its local devices and infrastructures to collect local metadata items and disseminates them into the monitoring overlay to be stored in a distributed fashion according to the metadata values. When a query is made for items specifying the range for certain attributes from any arbitrary node, a distributed search is initiated across the monitoring overlay. The primary objective of the query is to locate those nodes (LSCs) that store the requested metadata items. The tree structure with hierarchical ranges makes this query to be served efficiently. Metadata items with dynamic attributes higher than the bounded rate of update are handled via multicast. We explain the bound in Section 3.4.

3.2.2 Basic Working of Q-Tree

An example to understand how Q-Tree works is shown in Figure 3.1(a), where a monitoring overlay is constructed using the distributed LSCs. Let us consider that we want to monitor the camera frame rate. The possible value can be an integer between 1 and 20 frames per second (fps). As we see in the figure, each node in the monitoring overlay is assigned a range interval that specifies which items it stores (say, node C is given a range $(6, 9]$, that means C stores information of cameras with frame rate within that range). Each node also knows the entire range of its subtree, e.g., node C knows that cameras with frame rate within $(6, 20]$ are stored somewhere in its subtree. Now, a node B has a camera with frame rate 15 in its local site and it tries to push this information as a metadata item into the tree. The metadata item then traverses the tree and reaches the node E where the item should be stored, as it contains the target range. Suppose, G makes a query for this item (*i.e.*, camera with frame rate 15). This time the query gets propagated to the node E in the same way and E returns the item information. In some cases, node E needs to communicate to the original metadata source B to validate the staleness of the item before it delivers the result to the query initiator. We use the same strategy for answering the multi-attribute range queries. The detail of it is given in Section 3.3.4.

3.2.3 Q-Tree Architecture

Q-Tree is a middleware service that sits in between the OS and application layer inside the LSC of the participating users. All communications with the TI application are done through socket connection. We develop Q-Tree as a modular framework, shown in Figure 3.1(b). Functionalities of each modular components are given below.

Monitoring Topology Manager: The failure and churn of TI systems are fairly low, yet possible. The *topology manager* is responsible to construct and maintain the monitoring topology (constructed using distributed LSCs) during the session run-time. Each node in the tree maintains a *nodebuddy* list, a list of all nodes whom it monitors for detecting failures via a periodic *heartbeat* message. We discuss detail about the node join, leave and failure in Section 3.5.

Metadata Storage: The *metadata store* is responsible for storing the metadata in the local gateway node (running LSC). It collects the metadata of items of the local devices and components on demand basis and stores it in the *local storage* to fit into the overlay. Also, it stores the metadata of remote items of its own range in the *remote storage*. One of the main contributions of this paper is to show how nodes can do multi-attribute queries efficiently, hence how to assign hierarchical ranges to store remote metadata of data items on its own range to make the search efficient.

Hierarchical Range Allocator: Immersive users usually join the system at the very beginning. This may not be true for the non-immersive users. Nodes can join incrementally in the tree. The *hierarchical range allocator* assigns ranges to the monitoring nodes as they appear. As shown in Section 3.3.3, the range allocator assigns ranges to the monitoring overlay to construct a distributed storage of the metadata items.

Query Engine: As we explained earlier, queries are normally given in a high level description. It is the responsibility of the *query engine* to resolve the description into a single multi-attribute composite range query. One of the significant properties of Q-Tree is that it injects only a single query to the overlay for any size of a composite multi-attribute query without any preprocessing. Also, users can query with any form of boolean expressions without any transformation. Details are given in Section 3.3.4.

Metadata Router: The *metadata router* is responsible for efficiently routing the query messages and replying the result back in the aggregate manner. We support MAX, MIN, AVG, COUNT and SUM in-network aggregation. Section 3.3.4 gives details of how metadata router works to route the monitoring queries.

Load Balancer: We assign ranges to the nodes in the tree depending on the distribution of the attributes. But during the run time, the distribution can change and the load can be skewed. So, we use a *load balancer* similar to the Direct Neighbor Repeated [76] load-balancing approach. Each node periodically exchanges messages to its neighbor about its own load along with the heartbeat message and the load information is eventually propagated to the root. If the ratio of maximum load and minimum load in the system is greater than a threshold, the root initiates a load balancing algorithm where the highly loaded nodes transfer the load to its candidate range neighbors. We explain

the load balancing technique more elaborately in Section 3.6.

3.3 Session Monitoring Solution

3.3.1 Metadata Collection

LSC of each site monitors its local devices to collect local metadata items. We define a metadata item d as tuples of (attribute, value) pairs, d=$\{(a_1,v_1),(a_2,v_2), \cdots,(a_n, v_n),I\}$, where v_i is the value of attribute a_i and I is *item-Info* that contains information of the metadata originating site. For example, the monitoring information of a camera at site "teeve1.cs.xyz.edu" can be expressed using $\{$(*framerate*, 10), (*shutter*, 120), (*gain*, 133), (*zoom-level*, 12), (*3D-reconstruction time*, 10), (*avg. memory usage*, 78%), (*avg. CPU utilization*, 34%), *itemInfo*$\}$, where *itemInfo* = (node-info = camera-node, camera-id = 2, host-gateway = teeve1.cs.xyz.edu, OS = RHL, Kernel version = 2.2.6). Note that *itemInfo* is not an attribute about the device, rather provides environmental information where the device belongs to.

Although attribute values are usually continuous (sometimes, may be discrete) and can take wide range of values, we assume that all values can be normalized to $(0.0, 1.0]$. This can be easily done if the domain experts can somehow specify the minimum and maximum possible values for any attribute. Then, the simple equation $\overline{v} = \frac{v - v_{min}}{v_{max} - v_{min}}$ normalizes the value v to $(0.0, 1.0]$.

3.3.2 Monitoring Topology Construction

Q-Tree uses a hierarchical organization of LSC nodes in a tree-overlay, because tree provides in-network aggregation of query results. As a default overlay, Q-Tree constructs a k-MST, a degree bounded minimum spanning tree (DBMST) with no degree greater than k. k-MST's choice is due to the observation that in a tele-immersive system, participating nodes may have resource constraints to serve arbitrarily many other participants. k-MST limits the number of sites connected to a single node to be at most k. Q-Tree's query engine is, however, designed orthogonal to the control overlay and it can be mounted on top of any tree. Of course, since queries will be traversing along the tree, we wish to design a locality aware and latency optimal tree.

3.3.3 Hierarchical Range Assignment

Once the overlay is constructed, nodes are assigned with ranges. A range r is denoted as $r(l, h]$, where l and h are respectively the lower and higher limit of the range and $0 \leq l < h \leq 1$. v, the value of attribute (metadata) a, lies within r, i.e., $v \in r$, if and only if $l(r) < v \leq h(r)$. A range r_1 is contained within r_2, hence $r_1 \prec r_2$, if all values in r_1 also lie in r_2, *i.e.*, $l(r_2) \leq l(r_1) < h(r_1) \leq h(r_2)$. Two ranges r_1 and r_2 are said to be equal if they have the identical lower and higher limits, and are said to be consecutive if $h(r_1) = l(r_2) \vee h(r_2) = l(r_1)$. Two consecutive ranges can be

'unioned' to form a larger range like $r = r_1 \cup r_2$ and $r = (l(r_1), h(r_2)]$. We denote $|r| = h(r) - l(r)$ as the length of the range.

Let T be the rooted tree where $p(X)$ and $C(X)$ are the parent node and the set of child nodes of node X. Each node X is assigned with two ranges, *self-range* $\delta^a(X)$ and *subtree-range* $\Delta^a(X)$ for each attribute (metadata) a. Subtree-range $\Delta^a(X)$ specifies the range assigned to the entire subtree rooted at node X. A metadata item d with an attribute-value pair (a, v) is stored at node X if $v \in \delta^a(X)$. If $v \in \Delta^a(X)$, then metadata d is stored somewhere in the subtree rooted at node X. By definition, $\delta^a(X) \prec \Delta^a(X)$, i.e., self-range is always contained in the subtree range. The following four properties are hold by the ranges assigned to nodes for any attribute a:

- **Disjointness.** $\delta^a(X) \cap \delta^a(Y) = \emptyset$, all any pair of nodes X and Y.
- **Subtree range.** $\delta^a(X) \prec \Delta^a(X)$ and $\Delta^a(X) = \delta^a(X) \bigcup_{Y \in C(X)} \Delta(Y)$.
- **Hierarchy.** If X is an ancestor of Y, $\Delta^a(Y) \prec \Delta^a(X)$.
- **Entirety.** $\Delta^a(root(T)) = (0.0, 1.0]$, and $\bigcup_{X \in T} \delta^a(X) = (0.0, 1.0]$, where $root(T)$ is the root node of T.

The range assignment is initiated by root invoking *AssignRange*$_{root}(0.0, 1.0)$. Each individual node assigns range to itself and to nodes in its subtrees, similar to a preorder traversal of the tree. The algorithm for range assignment is given in Algorithm 1.

Input:
X: Node to which assignment is being made;
a: Attribute;
l, h: real numbers, range bound,

$\Delta^a(X) \leftarrow (l, h]$;
$\delta^a(X) \leftarrow (l, l + \frac{1}{N}]$;
$l \leftarrow l + \frac{1}{N}$;
for all c in $C(X)$ **do**
  $n_c \leftarrow c.size$; /* size of the subtree at child c */
  $h \leftarrow l + \frac{n_c}{N}$;
  $AssignRange_c(a, l, h)$;
  $l \leftarrow h$;
end for

Algorithm 1: Range assignment algorithm: $AssignRange_X(a, l, h)$

We assume that all values in the entire range $(0.0, 1.0]$ of an attribute are equally likely. So, each node gets a self-range of equal size $|\delta^a(X)| = \frac{1}{N}$ for each attribute. This ensures that each node stores nearly the same amount of data items. But if it happens that certain attribute follows some distribution other than uniform, the range should be partitioned depending on that distribution function (if known before-hand). For any given distribution function $F_a(v)$ for an attribute a, we need each node to get the equal number of data items to store, that is $P\{v \in \delta^a(X)\} = \frac{1}{N}$. Hence,

$P\{l < v \leq h\} = F_a(h) - F_a(l) = \frac{1}{N}$, *i.e.*, $h = F_a^{-1}(F_a(l) + \frac{1}{N})$, and for $\Delta_a(c)$, it would be $h = F_a^{-1}(F_a(l) + \frac{n_c}{N})$. Figure 3.2 shows examples of two range allocations for two different distributions. The first range of each node shows its own range and the second range indicates the subtree range. For example, node E in Figure 3.2(b) has its own range $(0.68, 0.74]$ and its subtree range is $(0.68, 1.0]$.

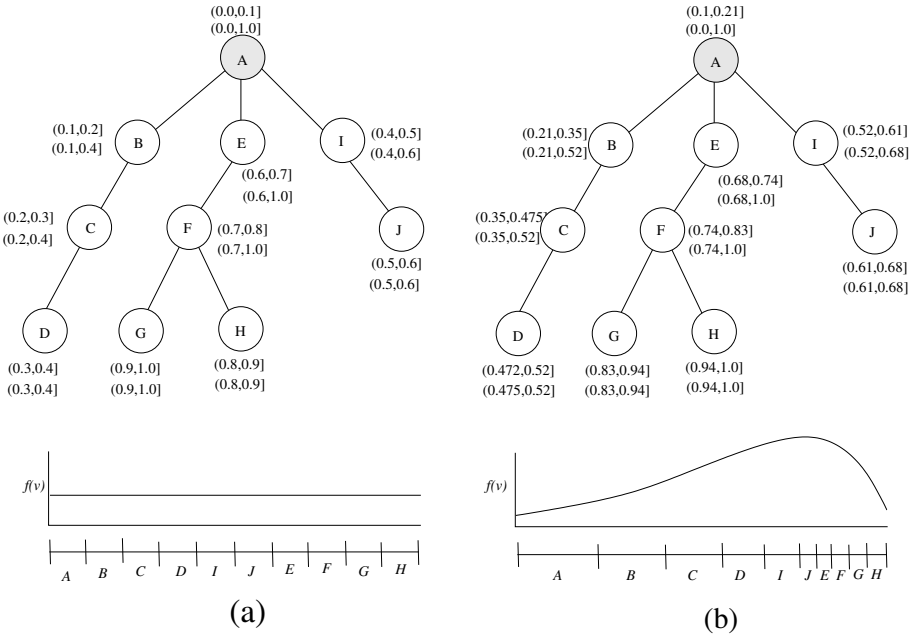


Figure 3.2: Range assignment (a) Uniform distribution, and (b) right skewed distribution.

If all attributes have identical distributions, then at a particular node X, self-ranges for all attributes become equal, *i.e.*, $\delta^{a_1}(X) = \delta^{a_2}(X) = \cdots = \delta^{a_l}(X)$. This is also true for the subtree-ranges. In that case, nodes require to keep only a single self-range and subtree-range for all attributes instead of ranges per attribute. Therefore, each node keeps $|C| + 2$ ranges: self-range and subtree-ranges for parent and $|C|$ number of children. If all attributes are treated differently, it becomes $L \times (|C| + 2)$ ranges for a system of total L attributes. In our current implementation, we, however, assume similar and uniform distribution for all attributes.

3.3.4 Q-Tree Query Engine

Once ranges are assigned for each attribute, the system becomes ready to fuse metadata items into the overlay and serve queries. A metadata item can be inserted as a collection of (attribute, value) or single (attribute, value) pair associated with *item-info*.

Inserting Metadata Item: As mentioned before, metadata items originating from the LSCs are fused in the overlay to be stored by remote LSCs. Whenever new metadata items are created or any attribute of an inserted item changes, they are fused in the overlay. A metadata item d is stored at a node X if any of d's attribute value lies

within the self-range of node X for that attribute. If any attribute value lies within the child's subtree range, the item is forwarded to that child. The same happens for the parent node. In Figure 3.3(a), we show how the insertion works for metadata item $d = [(a, 0.73), (b, 0.59), I]$.

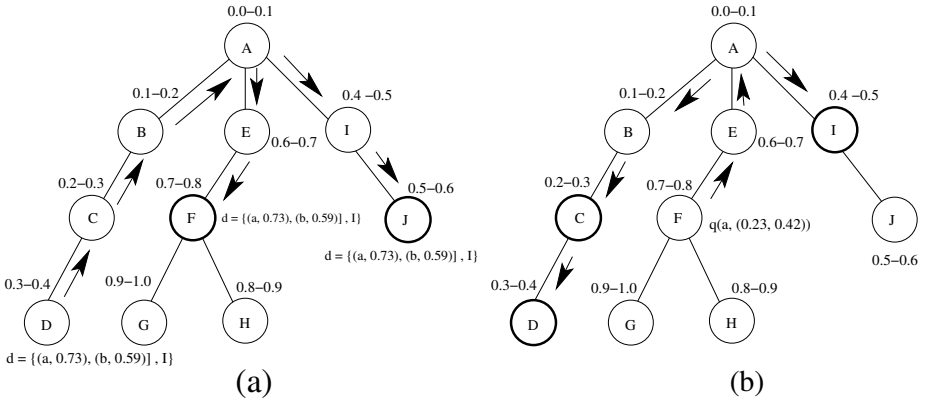


Figure 3.3: (a) D inserts $[(a, 0.73), (b, 0.59), I]$, the item is inserted at F and J. (b) F initiates query $q(a, (0.23, 0.42))$, query results are returned by C, D and I.

Answering Multi-attribute Query: Once metadata are fused, the overlay is now ready to serve queries. Q-Tree is especially designed for range queries with single or multiple attributes. The central session controller can originate a query by passing the query statement to any local session controller nodes in the overlay, which initiates a distributed search to locate nodes where metadata are stored.

Q-Tree looks for metadata items where the attribute value lies within the queried range. A range query with a single attribute is expressed as a predicate $q(a, r)$ that becomes true for a metadata item whose value v for attribute a is within r, *i.e.*, $l(r) < v \leq h(r)$. In that case, we say d *satisfies* q, denoted as $d \stackrel{s}{\Rightarrow} q$. The query propagates via the tree to reach the nodes where the requested items are stored. A node X contains candidate items for $q(a, r)$ if $\delta^a(x) \cap r \neq \emptyset$. Similar check ($\Delta^a(c) \cap r \neq \emptyset$) can be made to see whether any of its subtree contains the result. A camera with *framesize*=0.45, evaluates the query q(*framesize*, $(0.3, 0.5)$) as true, but one with *framesize*=0.7 evaluates it false. Figure 3.3(b) shows a query example where all values are normalized between 0 and 1.

A query can also be of type less-than ($<$) or greater-than ($>$). In that case, only one end of the range is given; another end of the range can be chosen as appropriate. The query for attributes $a \leq v$ (where $0 < v \leq 1$) is equivalent to $q(a, (0.0, v])$, where the query for $a > v$ is equivalent to $q(a, (v, 1.0])$. For equality (like, frame rate = 0.5) range becomes singular. This is handled by a special tag passed with the query statement. Note that, in the user-level, queries are given with real values in ranges, but Query Engine transforms such ranges in between 0 and 1.

One of the interesting aspects of Q-Tree is that it can handle complex range queries with multiple attributes as efficiently as it does for a single attribute range. An exam-

ple of a complex multi-attribute query can be 'find cameras with (frame rate within $(5, 20)$ OR frame size ≥ 1024) AND (shutter ≤ 60 OR gain > 123))'. Q-Tree handles this kind of complex queries by expressing the query as a composite query predicate. A composite query predicate $\mathcal{P}$ is a boolean expression that contains a set of single attribute query predicates joined by boolean operators. For example, $\mathcal{P} = q_1(a_1, r_1) \wedge q_2(a_2, r_2) \vee \neg q_3(a_3, r_3)$. In general, composite query predicate is expressed as $\mathcal{P} = q_1 \diamond q_2 \diamond \cdots \diamond q_m$, where $\diamond$ is any arbitrary boolean operator like $\wedge, \vee, \oplus$, and each q_i can be either q or $\neg q$. Therefore, $d \overset{s}{\Rightarrow} \mathcal{P} \equiv (d \overset{s}{\Rightarrow} q_1 \diamond d \overset{s}{\Rightarrow} q_2 \diamond \cdots d \overset{s}{\Rightarrow} q_m)$ or $d \overset{s}{\Rightarrow} q(a, r) \equiv \exists_{(a,v) \in d}(v \in r)$. Unlike other range query schemes, like Moara [43], Q-Tree does not require to express the predicate in some canonical form, rather any arbitrary boolean expression simply works. Algorithm 2 shows the Q-Tree query algorithm for any composition of multi-attribute metadata.

x: Query made at node x;
$\mathcal{P} : q_1(a_1, r_1) \diamond q_2(a_2, r_2) \diamond \cdots \diamond q_m(a_m, r_m)$;

$R \leftarrow \emptyset$;
$\Lambda = (\delta^{a_1}(x) \cap r_1 \neq \emptyset) \diamond (\delta^{a_2}(x) \cap r_1 \neq \emptyset) \diamond \cdots \diamond (\delta^{a_m}(x) \cap r_m \neq \emptyset)$;
if (Λ = True) **then**
  /* This node contains items that satisfy $\mathcal{P}$ */
  find item $d \in \mathcal{R}$ such that $d \overset{s}{\Rightarrow} \mathcal{P}$
  $R \leftarrow R \cup d$;
end if
for each child $c \in C(x)$ **do**
  $\Lambda_c = (\Delta^{a_1}(c) \cap r_1 \neq \emptyset) \diamond (\Delta^{a_2}(c) \cap r_2 \neq \emptyset) \cdots \diamond (\Delta^{a_m}(c) \cap r_m \neq \emptyset)$
  if (Λ_c = True, for a child c) **then**
    $R \leftarrow R \cup \mathit{Query}_c(\mathcal{P})$;
  end if
end for
if (If for any a_i in $\mathcal{P}$, $\Delta^{a_i}(x) \cup r_i \neq \emptyset$) **then**
  $R \leftarrow R \cup \mathit{Query}_{p(x)}(\mathcal{P})$;
end if
return R;

Algorithm 2: Q-Tree query algoritm: *Query$_x$(Predicate $\mathcal{P}$)*

Answering Aggregate Query: Sometimes an aggregated result over the items is requested rather than the list of items itself. For example, 'find MIN CPU utilization where memory $\leq$ 1GB', returns the minimum of all CPU utilizations of gateways that are equipped with more than 1GB memory. Q-Tree supports a set of aggregate functions, namely MIN, MAX, COUNT, SUM, and AVG. In that case, each node replies only a single value computed over the resultant items as defined by the requested aggregate function. Some aggregate functions cannot be applied on partial results, like DISTINCT COUNT or MEDIAN. In that case, entire result is accumulated before applying the aggregate function.

3.4 Handling Metadata Dynamics

Whenever a new metadata is inserted or attribute value of an inserted metadata is updated, they are simply fused in the overlay. Yet the old metadata items are not deleted immediately. Instead they are deleted *lazily* whenever they are attempted to be returned in response to a query. When a metadata is inserted or updated, the session monitor component sets timestamp on the item along with the original source of the item. If the item becomes older than some *freshness* threshold at the time when it tends to be returned against a query, the remote node contacts to the original source node to check the validity of the item. It is checked whether any attribute value for this item has been changed. If nothing is changed, the item stays at the remote node and the timestamp is updated. Otherwise the item is deleted.

Some attributes, like CPU utilization or available bandwidth change quite frequently and they generate frequent insertions into the monitoring overlay due to their updates. If they are not searched as frequently as they are updated, their insertions may incur substantial communication cost. In that case, items would preferably reside at the original node, and a simple multicast seems more efficient to serve range queries. To understand the relationship between the query cost for network communication, and the query rate of the metadata, we present an analysis below.

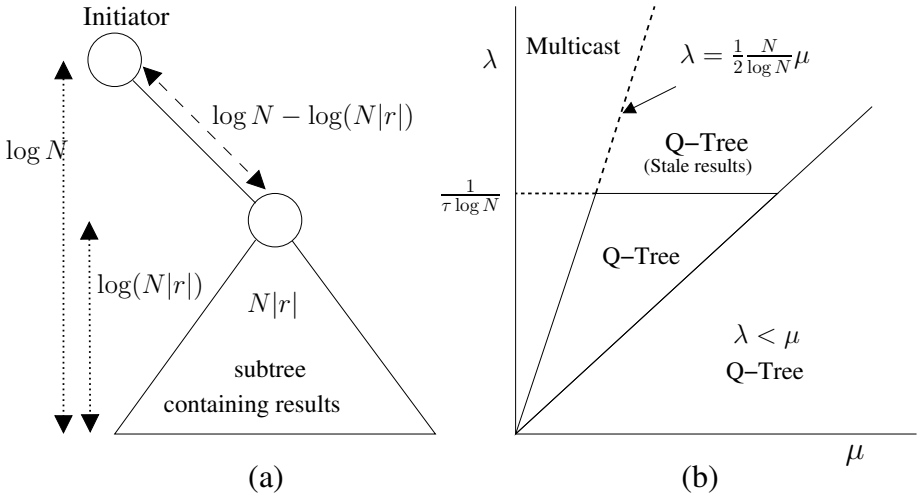


Figure 3.4: (a) Message cost for the query $q(a, r)$. (b) Operating zone of Q-Tree.

For a network of N nodes, we can verify that the message cost for multicast is $2N$. It simply follows from the fact that the query reaches to every node and returns back to the source. Let us find the message cost for the query $q(a, r)$. If the ranges are assigned uniformly over all nodes, then the number of nodes that contain items satisfying $q(a, r)$ is roughly $N|r|$. The query is pictured in Figure 3.4(a). The resulting items can be assumed to be confined in a subtree of size $N|r|$, which are returned by a multicast in that subtree. Yet the query has to reach that subtree to initiate the multicast inside. The multicast cost is $2N|r|$ and the cost to reach to that subtree

is roughly $\log N - \log(N|r|)$. So, total message cost for the query is $cost(|r|) = 2(\log N - \log(N|r|) + 2N|r| = 2(N|r| - \log|r|)$. We can check that $cost(1.0) \approx 2.N$, (the cost of multicast in the entire tree), and $cost(\frac{1}{N}) = 2\log N$, which is equal to the cost of query to a single node.

Let the value of attribute a change at rate λ and the range query on a is made at rate μ. Multicast cost is $2\mu N$, whereas in Q-Tree, the cost is the sum of insertion and query cost, that is $\lambda \log N + 2\mu(N|r| - \log|r|)$. Therefore, Q-Tree has smaller message cost than multicast as long as:

$$\lambda < 2 \times \frac{N(1-|r|) + \log|r|}{\log N}\mu$$

The above inequality gives the maximum rate at which attribute value can be changed when Q-Tree's message cost remains smaller than multicast. On average $|r| = \frac{1}{2}$ gives, $\lambda = \frac{N}{\log N}\mu$. For $N = 100$, $\mu = 1$/min, $\lambda = 50$/min. Again, if data changes faster than the time it takes to insert into the overlay, then the query may return stale data. This stale return can be avoided if $\frac{1}{\lambda} > \frac{1}{\tau}\log N$, *i.e.*, $\lambda < \frac{1}{\tau \log N}$, where τ is the average link latency. For $\tau = 200$ms, we have $\lambda = 150$/min. With increasing μ, allowed λ can be increased, but it cannot be larger than $\frac{1}{\tau \log N}$. Beyond that, Q-Tree switches to multicast for those data items. We find an operating zone for Q-Tree and multicast as shown in Figure 3.4(b).

3.5 Monitoring Overlay Maintenance

When a new node joins, the admin-gateway handles the registration and attaches it to an existing node considering its locality. The attached node becomes its parent. Then the parent node halves its self-range and assigns the higher range to that node.

Each node sends periodic heartbeat message to all of its neighbors (children and parent) to detect failure. Each node keeps a *nodebuddy* list, a list of nodes whom the node monitors for failure. In Q-Tree, children of a node are the nodebuddies of the parent, and the root is monitored by one of its child. Node can voluntarily leave or fail. In either case, the parent node creates a process representing the child until the child node reappears in the system. The parent node temporarily appends the child range with its own and acts as the child node in storing items and forwarding queries. In case of voluntary departure, the child transfers all items it was storing before it leaves. In failure, parent fetches all items that child was storing by making a query over the child range. To make this happen, each node keeps states (self-range, children list) of its nodebuddy members.

3.6 Load Balancing

In Q-Tree, we place all nodes in continuous range of a logical ring for load balancing purpose. The predecessor and successor of a node along the ring are the nodes that

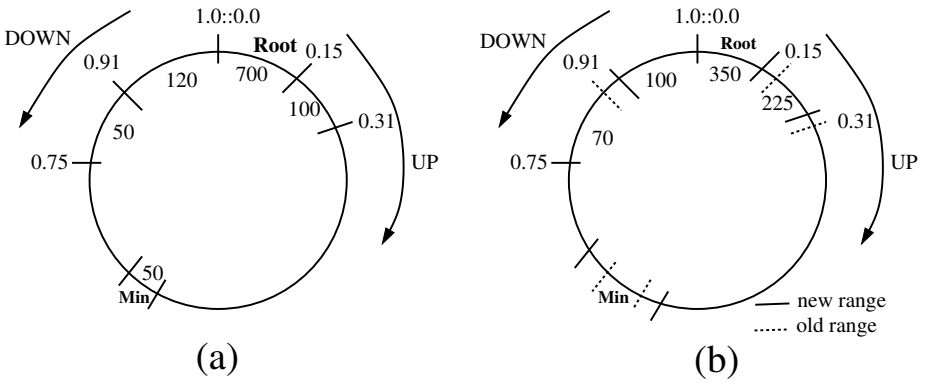


Figure 3.5: Load balancing for $LD = 2$. (a) Initial load, and (b) after one pass, load and ranges are adjusted.

have the subsequent ranges before and after the given node in the tree structure. We call them *range neighbors*. Note that this adjacency may not be related to the parent-child relation in the monitoring tree overlay. If a node wants to reduce its load, it needs to adjust its range with one of its range neighbors (predecessor or successor).

The load balancer works in several passes. Every node periodically reports the $minload$ and $maxload$ of its subtree to its parent via the heartbeat messages. Thus the root has the information about $minload$ and $maxload$ of the whole system. If there is a load imbalance, that is $maxload > LD \times minload$, root initiates the load balancing algorithm in two directions via its successor (UP) and predecessor (DOWN), where LD defines the imbalance factor. If any node in the path finds its $load > LD \times minload$, it makes $load = \max(\frac{load}{2}, LD \times minload)$ and adjusts its range and data items accordingly. It adjusts its range and transfers the excessive data items to its successor (if its in UP direction) or predecessor. When the balancing reaches the $minload$ node, it notifies the root and the current pass is done. After sometime, root starts another pass. We use, $LD = 2$ so that no nodes are allowed to store data items more than double of other nodes. Figure 3.5 shows as an example of load balancing operation. Figure 3.5(a) shows a initial range assignment in Q-Tree which causes overloading at the root node. We run our load balancing algorithm with $LD = 2$. The balanced range condition is shown in Figure 3.5(b).

3.7 Experimental Evaluation of Q-Tree

3.7.1 Experimental Setup

We simulate Q-Tree in an emulated network setup on top of a discrete network event simulator coded in Java. We emulate a 'virtual' network with node-to-node latencies obtained from 4 hours PlanetLab traces [77] which consists of 250 distinct nodes. This trace gives us the connectivity information and rtt delays among the nodes. We use this information to simulate query plane for TI systems. As an overlay choice, we consider MST and k-MST (degree bounded MST). We are interested in mainly three

performance metrics: query latency, communication cost, and overhead in metadata maintenance. We consider queries of three kinds, namely equal-to (EQ), multi-attribute composite range queries (R), and multi-attribute multicast queries (M). EQ represents query specified by an '$attr = value$', for example 'cameras with *framerate*=20', R specifies limits on attribute values, such as 'cameras with *frame rate* between $(10, 15)$ and *gain* > 100'. Range queries (for both static and dynamic attributes) may also optionally be accompanied with any of the aggregate functions like MIN, MAX, COUNT, SUM and AVG, for example 'what is the MAX *framerate* in current TI session'. For each run, we build the overlay by taking nodes randomly from the traces and simulate the same event for 100 instances and take their average. Unless otherwise stated, each site has 10 cameras and each camera generates 10 attributes to monitor.

3.7.2 Monitoring and Query Performance

Figure 3.6(a) shows the latency of different queries for different number of nodes in the system. Here the overlay choice is MST. As we see in the plot, the average latency for 'equal-to' (EQ) queries is fairly small compared to the other queries in the system. This is reasonable because EQ only searches for a single node in a specific range. 'Multi-attribute range' (R) queries are more common in TI system and due to the hierarchical range assignment, those queries can be served within less than a second on the average even with 250 PlanetLab nodes. Another reason for getting lower latency is due to the 'bottom-up' approach in overlay construction which ensures a latency-optimal tree and thus further reduces the latency for multicast queries. In Figure 3.6(b), we show the message count of these three types of query. Message count includes the number of messages used to propagate the query and to reply back the result to the originating node. For multicast, the message count is always the twice the number of nodes in the system because of the aggregated reply. As we see in the figure, for EQ and R, the message count changes very slowly with the number of nodes.

In Figure 3.6(c), we present the impact of degree bound on overlay for range queries. We compare the latency of multi-attribute multicast queries with MAX function for MST, and several values of k. The result is reasonable: when $k = 2$, each node is allowed to have only one child in the tree which makes the tree deep enough and thus increases the latency. With the increase of the k value the latency decreases (*i.e.*, the performance increases) and gives the maximum performance in case of MST. However, there is a trade-off between the message overhead per node and the latency, for the choice of different k values. For MST, the standard deviation of message count per node per query is fairly high, because of wide variation of degree counts of nodes. The result is shown in Figure 3.6(d). When we bound the degree by k, the message is more evenly distributed among the nodes. So, we leave the choice of k on the administrator depending on the TI system characteristics.

Handling dynamism is an important contribution in our paper. Recall that data items change at rate λ and query is made at rate μ. In Section 3.4 we give a bound on λ. To validate the bound, we experiment with $N = 100$ and $\mu = 1$query/minute and change

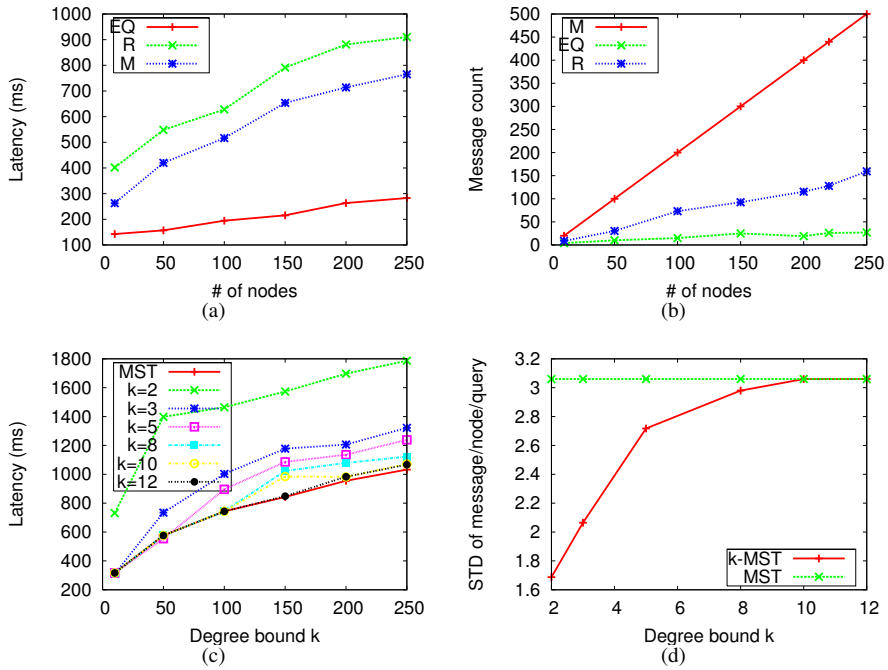


Figure 3.6: (a) Latency (MST), (b) Message count (MST). (c) Latency for different k with 100 nodes. (d) Standard deviation of message per node per query for different k and 100 nodes.

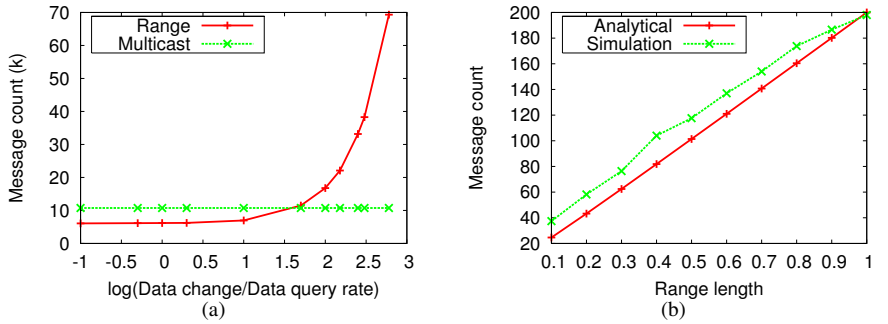


Figure 3.7: (a) Message overhead for different rates of update. (b) Message cost for different range intervals.

the value of λ as some multiple of μ. The message overhead for an hour of inserting data items at rate λ and the range queries at rate μ is shown in Figure 3.7(a). The straight line shows the message overhead due to the multicast. Two curves intersect at $\log \frac{\lambda}{mu} = 1.65, \lambda \approx 45\mu$ that is pretty close to the bound $\lambda = \frac{N}{\log N}\mu = 50\mu$. When λ exceeds the bound, multicast requires less messages. In that case, those data items are not inserted into the overlay. Earlier we showed that to answer $q(a, r)$ the number of nodes visited is $N|r| + \log(|r|)$. In Figure 3.7(b) we evaluate it. The upper line indicates the analytical result while the lower one is the simulation result. The theoretical result coincides with the experimental values.

The next experiment shows the scalability of Q-Tree with respect to the number of

local devices at each TI site. We consider each local device with 10 different attributes in 100 different sites. Figure 3.8(a) shows the latency of insertion into the overlay. We measure the latency in 3 cases: 1) inserting all data items instantaneously from each site, 2) inserting 500 data items per second from each site and 3) inserting 1000 data items per second. X-axis shows the number of devices connected at each site and Y-axis shows the latency in millisecond. Even with significant insertion rate, the insertion latency remains below 1sec.

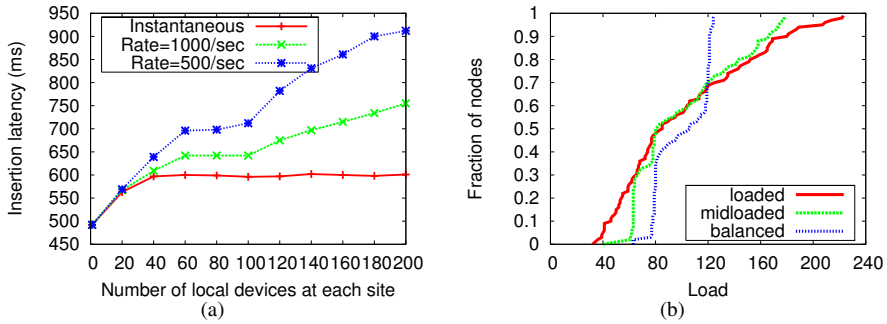


Figure 3.8: (a) Latency of data item insertion/update (10 sites). (b) Load distribution across nodes (LSCs).

To measure the performance of our load balancing algorithm, we run Q-Tree with the skewed distribution (beta(4,2)) of the metadata values for 100 nodes in the system. We define γ as the current ratio of maximum and minimum load of the system. We set γ to 6.968 at the beginning. The load balancing algorithm terminates when the ratio goes below 2 (i.e., when $LD \leq 2$. The system is 'loaded' if $\gamma > LD$. Figure 3.8(b) shows the distribution of nodes with respect to the current load at different passes of the algorithm. At the beginning, $\gamma = 6.0968$ and the algorithm terminates when $\gamma = 1.98 < 2$.

3.8 Conclusion

Tele-immersive interactive systems present new challenges in the distributed systems area. We have designed and evaluated Q-Tree, a multi-attribute query framework for querying dynamic TI systems. Our performance results show that our system scales with the number of local devices and data items and allows sufficient attribute churns. Any complex multi-attribute range query can be served in low latency and minimum overhead. Beyond TI, any system that needs multi-attribute range queries in time-sensitive, light-weight and efficient manner, will be benefited from Q-Tree. In our dissertation, Q-Tree works as a monitoring solution for taking decisions for immersive and non-immersive session configuration.

4 TI Activity Driven QoS Optimization

4.1 Introduction

As mentioned in Section 5.3.2, 3D tele-immersive systems are used for various activities and each activity defines its own session parameters to guarantee satisfaction of immersive users. The session parameters include cyber-physical QoS profiles such as minimum values of QoS parameters and a priority relationships among them. In this dissertation, we do not focus on constructing QoS profiles for TI activities, rather we focus on given such profiles how to guarantee them. TI participants we consider in this chapter are solely immersive users since the dependency of activity on the non-immersive users are currently not clear.

Our overarching goal is to address the QoS allocation problem as a QoS optimization process in the session management pipeline as shown in Figure 1.5. The goal of QoS optimization is to construct a 3DTI content distribution graph considering available resources, content demands and QoS specifications (minimum quality bounds and priorities) of the on-going activities. To this extent, we propose *Optimized Immersive Session Management* or OSM algorithm, which formalizes the QoS optimization in multi-site multi-stream topology construction as a priority-based *multi-objective optimization (MOOP)* problem. However, the constraint-based multi-objective optimization in a network routing is an NP-complete problem [78][79] and hence, we need a heuristics-based solution. In OSM, we chose an evolutionary algorithm.

Evolutionary algorithms (EAs) are well-known heuristics to solve MOOP. They use an iterative approach and therefore, may include a large latency in the search of an optimal multi-stream content distribution topology. However, using a careful selection of initial solution (e.g., multi-stream multi-site overlay graph satisfying minimum QoS quality and resource constraints) in the search space, and allowing topology diversities in the optimality search, we show a novel usage of EA in 3DTI that ensures fast convergence towards optimized QoS allocation. Moreover, to mask the evolutionary optimization latency and to improve response time in session adaptation, we extend OSM as a two-step process (Section 4.7). In the first step, an instantaneous non-optimal solution is installed to meet the demand of the session update request while the optimal solution is computed in background. The second step installs the optimal solution.

We implement a prototype of the OSM architecture and show the feasibility of 3DTI session optimization using bandwidth and end-to-end delay traces from PlanetLab [39] nodes. From the experiments, we show that the evolutionary session optimization provides up to 50% improvements in the allocation of desired QoS parameters, compared

to the current content distribution solutions in the 3DTI space. The session adaptation using one step optimization process takes about 2-3 seconds. However, using the two-phase approach, the session adaptation latency reduces to less than 300ms. We also perform a set of subjective studies under different 3DTI activities with 23 participants to justify the performance of OSM (Section 4.8.5).

4.2 QoS Parameter and TI Activity Model

4.2.1 QoS Parameter Model

We consider two types of QoS parameters (or metadata) in the 3DTI space: *controllable QoS* (cQoS) metadata and *derived QoS* (dQoS) metadata. An application can control cQoS metadata by configuring the content distribution topology and application parameters, for example, end-to-end delay (EED), bit rate[1] of a stream s (R_s), number of video streams to accept from each remote site in order of priority (NVS) and number of total streams to accept from each remote site in order of priority (NSC). If x_i indicates the i^{th} cQoS metadata, the set can be represented as $X = \{x_1, x_2, \cdots, x_m\}$ for m number of cQoS metadata.

There are some QoS metadata over which the session layer does not have any direct control, for example, 3D rendering time, application frame size, participants view orientation and so on. We call them dQoS metadata. They are important for identifying the system health of running sesssions.

4.2.2 TI Activity Model

Activities in the 3DTI space can be defined using the movement-based characteristics of the participants. Each activity α defines a minimum quality bound min_x^α for each cQoS parameters $x \in X$ and assigns a priority p_x^α to it. For m number of cQoS parameters, $p_x^\alpha = m$ indicates the highest priority and $p_x^\alpha = 1$ indicates the lowest priority value. The maximum quality bound max_x^α of the cQoS parameter x depends on the application design (e.g., video coding) and the device characteristics (e.g., maximum frame rate), and are non-variant across activities. The value of min_x^α and p_x^α can be obtained using a systematic subjective and objective evaluation for that activity [18].

4.3 Motivating Study

Using TEEVE [4], we have explored a wide variety of interactive activities and qualitatively measured cQoS parameters that influence the final users' QoE. First, we define the list of cQoS parameters (Section 4.3.1) and TI activities (Section 4.3.2) we used in our study. Section 4.3.3 defines evaluation method to identify the impact of activity

[1]The stream bit rate can be defined using the media quality (such as the color-plus-depth level-of-detail for 3D video [19] and perceptual evaluation of speech quality [80] for audio) and the media frame rate.

on QoS and users' QoE. Finally, we summarize our observation in Section 4.3.4 that motivates the development of OSM.

4.3.1 cQoS Parameters

For the motivating study, we consider a set of cQoS parameters from three major categories: *video cQoS*, *audio cQoS*, and *cross-media cQoS*. Table 4.1 shows the cQoS parameters and their definitions.

cQoS parameters	**Definition**
Video Quality (VQ)	Spatial video frame resolution, measured in number of pixels per frame, bits per pixel, PSNR (Peak Signal-to-Noise Ratio), and Color-to-Depth Level-of-Detail (CZLoD) [19].
Number of Video Stream (NVS)	Number of video streams in a bundle of streams.
End-to-End Delay (EED)	Time interval between when a media frame is captured and when it is displayed. It is considered for both audio (aEED) and video (vEED) media.
Video Frame Rate (VFR)	Application frame rate of a video stream.
Inter-stream skew (ISS)	Skew between streams of similar modality. Currently, we measure ISS among video streams.
AV Synchronization Skew (AVSS)	Perceptual skew between correlated audio and video frames.
Number of Sensory Channel (NSC)	Number of sensory devices used to construct immersive experiences.
Audio Quality: PESQ (AQ)	Quality of an audio signal as defined by the standard ITU-T P.862.
Audio Sample Rate (ASR)	Application frame rate of an audio stream.

Table 4.1: Cyber cQoS Metrics & Definition

4.3.2 3DTI Activity

We consider two activities to perform our motivating study. In the conversation activity, two participants talk to each other by being in the virtual world. To allow high-fidelity speech communication, we equip users with wireless/wired headsets with a microphone input. The wide-band speech signals are encoded with 44kbps data rate, using the wide-band Speex library. A 4-channel microphone array is an add-on capability to capture the ambient sound, which is encoded using Advanced Audio Coding (AAC). A passive stereo (as we mentioned in Section 1.1.2) was used at the 3D cameras to capture participants' 3D images.

In the virtual lightsaber dual, two players fight with each other in the virtual world using physical swords. To engage in virtual fencing, the following steps are taken: 1) participants wear lab coats with colored patches on them, 2) each participant uses the light-saber to hit the opponent's color patches in the virtual space as much and as fast as possible in order to gain points, and (3) when a hit occurs, the sword and the coat patches in the virtual space turn blue and the participant, making the hit, gets points. Also the participant, who gets hit, feels a haptic feedback through vibration and

lightning of his sword.

4.3.3 Evaluation Method

ITU (International Telecommunication Union) has prescribed several recommendations for evaluating perceptual quality of video-conferencing systems, which serve as useful guidelines for tele-immersion. But unfortunately, little is understood about the impact of different QoS configurations on different TI activities in terms of QoE. This motivates us to perform our own evaluations. The evaluation methodology involves three steps: (1) objective evaluation of QoS, (2) subjective evaluation of QoE and (3) finding correlations between QoS and QoE measurements.

Objective evaluation of QoS is performed by active monitoring using Q-Tree (Chapter 3). During and/or after activities, subjective evaluation of QoE is performed with human participants [21][19]. This evaluation utilizes methods that record self-reported responses of users, such as their perception of the video quality and their concentration level during activities. In TEEVE, we use questionnaires and comments from participants as subjective methods. Finally, we correlate objective QoS measurements with subjective QoE evaluation, for example, using comparative methods to find functional relations between user experiences and QoS configurations and resource allocation.

4.3.4 Observation

From the above experiments, we conclude that the importance of QoS metrics across different TI activities varies. For the conversation activity, the AQ is very important due to the dominant auditory and conversational nature of the activity. Therefore, the importance of PESQ and ASR are high. However, due to no or limited movement during the conversation, motion jerkiness at a reduced VFR is less noticeable. Another crucial QoS parameter for conversational activity is AVSS. A tight synchronization requires low skew (less than 160 ms) of video ahead of audio, and medium to high spatial resolution around lips/face. The reason is that the talk-spurt durations in the TI conversational activity are generally short, so lip skew at the end of an utterance is more noticeable. However, the EED tolerance level in conversation is medium (less than 400 ms). NVS is and NSC are less important, however; the audio is considered more important than video. Since only one prominent video streams are acceptable in the conversation activity, ISS is not important.

For the virtual lightsaber activity, the most important metric is the EED. It should be less than 100 ms. No one wants to lose in virtual fencing due to the presence of noticeable delays in the system. Without low and bounded EEDs, it is very hard to create a synchronized performance. Similar reasoning is applicable for ID. Unlike fine conversation, the lightsaber activity does not require high quality, but it requires a medium resolution for specific parts of the body (e.g., both upper and lower body of the image) and the corresponding VFR must be high (greater than 14 fps). The AQ is less crucial and so is the AVSS, since participants are closely engaged in visually dominant activities. Though the audio-video synchronization skew is not crucial, the

value of ISS for video streams is very important since a large inter-stream skew may create inconsistent views (e.g., upper body can shift, compared to the lower body).

Therefore, based on the above observations, TI activity defines two important cyber-physical QoS specifications in the session parameters: 1) minimum acceptable bound of the cQoS parameters, and 2) importance of each cQoS parameter.

4.4 3DTI Adaptive Session Optimization Framework

At the beginning of this section, we explain the necessity of an adaptive session optimization in the 3DTI session layer. At the end, we provide an architectural overview for the 3DTI session management.

4.4.1 Design Space

Why do we need session adaptations? 3DTI sessions are very dynamic since participants may change views or network resources may change over time. The goal of the session adaptation is to address these changes by modifying the cQoS parameters in the multi-stream content distribution topology. However, the configuration (at session initiation) and re-configuration (at session run-time) of the cQoS parameter during the adaptation process are dependent on the underlying TI activities.

When do we trigger session adaptations? We assume that the activity is selected at session initiation. The view change of the participants during session run-time is detected using the stream differentiation technique proposed in [4]. Resource modification is identified by running Q-Tree monitoring oracle at the participating sites. The content and resource modifications during the session run-time are called session updates. A session adaptation is triggered when the system experiences a session update.

What does happen during session adaptations? When a session update is triggered, 3DTI session *states* are modified; hence, the requirements on the cQoS parameters are updated. Maintaining minimum quality bounds of cQoS parameters in the updated 3DTI session only guarantees acceptable QoE to the participants. However, to maximize the perceivable QoE with maximum utilization of the resources, cQoS optimization is required. Since different TI activities put different priorities in the cQoS optimization, a prioritized cQoS optimization is needed in the adaptation process. Therefore, we formally define the 3DTI adaptive session optimization as the process of constructing a content distribution graph for multi-site multi-stream 3DTI contents (triggered by a session update) that optimizes cQoS values considering priorities subject to their minimum quality bounds and the availability of network resources. The pipeline of the 3DTI adaptive session optimization is shown in Figure 4.1.

In this paper, we only consider the optimization step in the 3DTI session adaptation process. The 3DTI session optimization is challenging because of two reasons. First, the optimization process needs to consider optimization of cQoS parameters in the order of the activity-defined priorities, and second, the process should maintain the

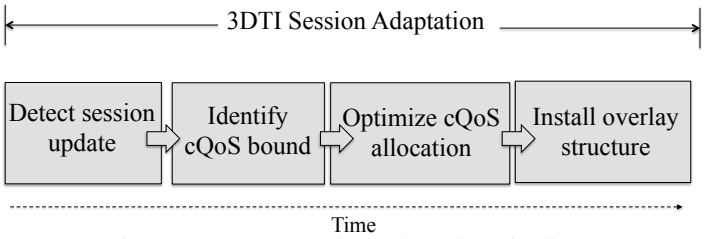


Figure 4.1: 3DTI session adaptation pipeline.

minimum quality constraints defined by the activity while constructing a multi-site and interest-based multi-stream content distribution graph. Both of the challenges are NP-complete. Below, we present our OSM architecture that focuses on solving this session adaptation problem.

4.4.2 Optimized Session Management Architecture

OSM uses the same architecture as shown in Figure 1.6 and manages session using the global session controller (GSC) and local session controller (LSC). Each participant gateway maintains a LSC. A global view of participating peers and networking infrastructures are maintained at the GSC.

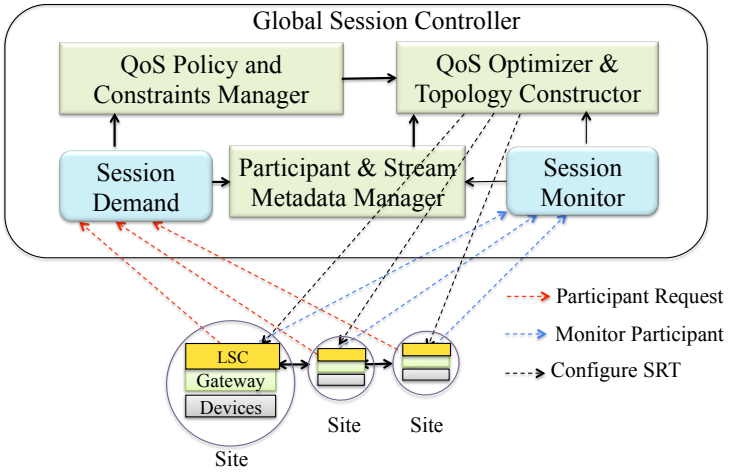


Figure 4.2: Components of Global Session Controller

Global Session Controller: The GSC is responsible for adaptive session optimization that uses the pipeline processes shown in Figure 4.2. It contains four components: 1) *session monitor* that periodically (few seconds) monitors the network resources, device availability and end-to-end delay of streams at each participating site using Q-Tree, 2) *participant and stream metadata manager* that manages participants information, detects participants' views and defines stream priorities, 3) *QoS policy and constraint manager* that stores an offline profiling of minimum bounds and priorities of cQoS parameters for each 3DTI activity, and 4) *QoS optimizer and topology constructor* that computes the optimized multi-stream content distribution graph and constructs a distribution topology using the inputs from the session monitor, stream metadata

manager and cQoS profiler. The topology is represented by session routing table (SRT) with three fields:

- **Match Fields: SRT$_{match}$ (53 bits).** It uniquely defines a stream in a running immersive session. It is a composition of UDP destination port (16 bits destination port), gateway ID of the stream residing site (32 bits IP addresses) and stream ID (5 bits monotonically increasing ID which is unique inside a site).
- **Forwarding Actions: SRT$_{action}$ ($\times$32 bits).** It defines a list of remote gateways to whom the matched stream (matched with SRT$_{match}$) needs to be forwarded.
- **Forwarding Rates: SRT$_{rate}$ ($\times$8 bits).** It defines a list of rates to be used for each forwarding path based on the QoS allocation defined by the optimization function.
- **Dirty Bits: SRT$_{dirty}$ ($\times$1 bit).** It indicates whether an entry has been modified or not compared to the last update.

Local Session Controller: The LSC is responsible for maintaining the policy (e.g., enforcing the constructed distribution topology) at the participating sites using the SRTs. It also helps the session monitor component at the GSC with the collection of metadata about participants' current views and local network conditions. To maintain the multi-streaming topology, the LSC maintains a *session routing table* at each gateway, which defines stream forwarding paths.

Data Plane: The data plane in the gateways simply considers the SRT installed in the local session controller and forwards streams according to the forwarding addresses and forwarding rates. As we discussed before, the rate for the audio and video streams can be adjusted by changing the application frame rate and/ or the media quality in the media encoding algorithm. We consider this configuration as the data plane responsibility and leave it out of the current scope.

4.5 Session Optimization

4.5.1 Problem Formulation

Input: A 3DTI system with N participants can be represented as a **complete graph** $G = (V, E)$, where V= $\{v_i\}$ is the set of gateways and E=$\{e_{ij}\}$ is the connecting network path from gateway v_i to v_j for $1 \leq i, j \leq N$ and $i \neq j$. Each vertex (site gateway) v is defined by a set of output streams o_v it generates, a set of request streams r_v it demands, an inbound bandwidth capacity C^v_{ibw} and an outbound bandwidth capacity C^v_{obw}. Each edge e_{ij} is defined by one way end-to-end delay d_{ij} between vertex v_i and vertex v_j. For ease of understanding, we use s^v_i to represent the i^{th} stream generated by vertex $v \in V$. Also, we assume that the set r_v is ordered in the descending order of stream priority (mentioned in Section 1.1.2) and S_{all} is the set of all streams available

in the system, i.e., $S_{all} = \{S_j^v\}, \forall v \in V$ and $1 \leq j \leq |o_v|$. For the current activity α, the priorities of the m cQoS parameters are defined as $x_1 > x_2 > x_3 > \cdots > x_m$.

Optimization Goal: The goal of session optimization is to construct a directional multi-stream and multi-site 3DTI content distribution graph G^{opt}=(V^{opt}, E^{opt}), where V^{opt}=V and $E^{opt} \subseteq E$. Each directional edge is associated with a stream and a rate, and each vertex is defined by an inbound and outbound bandwidth allocation for activity α that optimizes $x \in X$ in order of their priorities subject to the following constraints:

- **Resource (Bandwidth) Constraints:** If ibw_v and obw_v are the inbound and outbound bandwidth allocations, respectively, for vertex $v \in V^{opt}$ in graph G^{opt}, then $\forall v \in V^{opt}$, $ibw_v \leq C_{ibw}^v$ and $obw_v \leq C_{obw}^v$,
- **cQoS Constraints:** If min_x^α is the minimum quality (e.g., maximum delay, minimum rate) bound for parameter x, then $\forall x \in X$, $x < | > | = min_x^\alpha$ (e.g., maximum delay, minimum rate).

This session optimization problem represents a priority-based multi-objective optimization (MOOP) problem [78]. However, solving MOOP for a network graph even for a single source and destination with a single stream is NP-complete, in fact it is considered as an intractable problem for large networked systems [81][79]. Heuristics for solving the MOOP problem have been proposed since the constraint-based routing algorithms were discovered; however they are limited, even missing [46].

4.5.2 Priority-based Multi-objective Session Optimization

Evolutionary algorithms (EAs) have emerged as powerful heuristic tools for solving NP-complete and NP-hard problems and have received a great attention because of their ability for solving multi-objective optimizations in many areas such as network routing [78][82] and real-time playout scheduling [83]. We adopt genetic algorithm (GA) (a genre of EA) as a heuristic algorithm to solve the priority-based multi-objective optimization problem for 3DTI multi-site and multi-stream content distribution.

Genetic algorithms have derived their inspiration from the process of natural evolution and represent an iterative procedure of applying basic genetic operators over the candidate solutions of multi-stream multi-site content distribution graphs[2]. The genetic operators are: *selection*, *crossover* and *mutation*. The process starts with a set of initial solutions as the candidate solutions and then applies genetic operator on them in an iterative fashion to find an optimal solution. The following subsections describe these operators and the associated genetic algorithm in the immersive 3DTI space.

A. Encoding

Instead of binary encoding, we use a real parameter GA. A global solution graph (called *chromosome*) for multi-stream multi-site content distribution can be broken down into

[2]We represent candidate solutions as G'=(V', E'), G''=(V'', E''), G'''=(V''', E''') generated in the search for G^{opt}. On the other hand, G=(V, E) represents the complete multi-stream connectivity graph of the 3DTI environment, i.e., V''=V'''=V, and $\{E', E'', E'''\} \subseteq E$.

individual subgraphs representing the distribution of each stream in the system. For example, G'_S=(V'_S, E'_S) defines the solution subgraph for the distribution of stream S in G' (i.e., $G'_s \subseteq G'$). The chromosome (or global distribution graph) G' representing the distribution of all streams $S_i \in S_{all}$ is represented by $G'_{S_1}|G'_{S_2}|\cdots|G'_{S_k}$, where k is the number of total streams available in the current 3DTI session. If a stream S is dropped in a 3DTI session, an empty subgraph is represented by G'_S, where E'_S=ϕ.

An example of chromosome encoding is shown in Figure 4.3 with 3 sites (vertices are represented as A, B and C); each generating 2 streams (represented as S_1^A, S_2^A, S_1^B, S_2^B, S_1^C and S_2^C, respectively). In practice, the number of streams generated from each site can be much higher. Suppose, the request sets from the participants are, $r_A = \{S_1^B, S_1^C, S_2^B, S_2^C\}$, $r_B = \{S_1^C, S_1^A, S_2^C, S_2^A\}$ and $r_C = \{S_1^A, S_1^B, S_2^A, S_2^B\}$, where streams are listed in-order of their priorities. A sample multi-stream multi-site distribution graph (G') is shown in Figure 4.3 (a) and the corresponding subgraphs for individual streams are shown in Figure 4.3(b-g). The chromosome representing G' can be encoded as : **chromosome (c) =** $G'_{S_1^A}|G'_{S_2^A}|G'_{S_1^B}|G'_{S_2^B}|G'_{S_1^C}|G'_{S_2^C}$.

B. Initialization of Solution

The initialization of solution is very important for GA to ensure faster convergence towards the near optimal solution. To keep the search space and convergence time bounded (required in the interactive 3DTI space), we only consider *valid* chromosomes (i.e., valid content distribution graphs) in each iteration of the algorithm. A valid chromosome is defined as a multi-site, multi-stream distribution graph, where each cQoS metadata maintains at least the *minimum quality* subject to the resource (bandwidth) availability. However, the minimum quality problem for finding multicast trees subject to the bandwidth constraint is also an NP-complete problem [84] [85]. To construct an heuristic-based solution for finding the initial set of chromosomes with minimum quality, we use ViewCast [4]. To understand how ViewCast generates the initial solutions, we first define the minimum quality problem as follows.

Minimum Quality Problem: Construct a multi-stream multi-site 3D content distribution graph G'=(V',E') for α, that

1. satisfy $\forall x \in X,\ \ x < | > | = min_x^{\alpha}$,
2. subject to $\forall v \in V',\ \ ibw_v \leq C_{ibw}^v$ and $obw_v \leq C_{obw}^v$

Minimum Quality Guarantee Using ViewCast: Suppose, we need to forward a stream S to a node v in the process of constructing a global distribution graph as an initial solution. V'_S is the set of vertices currently holding stream s due to the current distribution topology. The relation $v_i \in V'_S$ is true if one of the following two conditions is satisfied: 1) v_i is the source of S (i.e., $s \in o_{v_i}$), or 2) v_i receives S via previously assigned network paths in the graph. Therefore, all $v_i \in V'_S$ are the candidates for forwarding streams S to v. We randomly select a vertex $v_i \in V'_S$ provided that both v_i and v have available bandwidth to satisfy the minimum bit rate of S and the establishment of the path from v_i to v does not violate the end-to-end delay. Similarly, the distribution graphs are constructed for all requested streams.

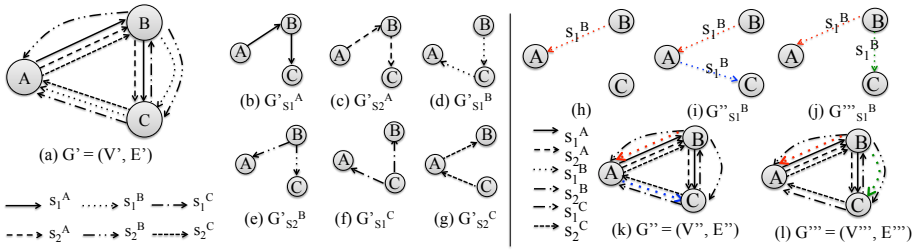


Figure 4.3: (a) Example of a multi-site multi-stream distribution graph (chromosome), (b)-(g) multi-site distribution subgraphs for individual stream (components of chromosome), (h) Edge e_{BA} is already assigned, i.e., $V_{S_1^B} = A, B$, (i) subgraph $G''_{S_1^B}$ is generated if vertex C gets S_1^B from vertex A, and (j) subgraph $G'''_{S_1^B}$ is generated if vertex C gets S_1^B from vertex B, (k) distribution graph G'' using subgraph $G''_{S_1^B}$, and (l) distribution graph G''' using subgraph $G'''_{S_1^B}$.

Example: Let us consider the 3DTI setup discussed in Section 4.5.2. The cQoS parameters we use here are R_S ($\forall S \in S_{all}$), EED, NVS and NSC. The process of constructing the distribution overlay in ViewCast is iterated from the highest priority stream to the lowest priority stream for each vertex for a given vertex order. If we consider a vertex order of $\{A, B, C\}$, the ViewCast constructs a distribution subgraphs for streams in the following order $\{S_1^B, S_1^C, S_1^A, S_2^B, S_2^C, S_2^A\}$. If we consider vertex order of $\{B, C, A\}$, then the stream order to construct subgraphs is $\{S_1^C, S_1^A, S_1^B, S_2^C, S_2^A, S_2^B\}$. Figure 4.3(h)-(l) show the construction of the distribution graph for stream S_1^B. Suppose, the ViewCast already assigns an edge e_{BA} to satisfy the demand of S_1^B to vertex A (Figure 4.3(h)). To satisfy the request for vertex C, we can get S_1^B from either vertex A (Figure 4.3(i)), or vertex B(Figure 4.3(j)). If both paths e_{BC} and e_{AC} have available bandwidth higher than the minimum value of $R_{S_1^B}$ and they do not violate the minimum bound of EED, any of these two paths can be selected randomly. However, if neither of them satisfies the bandwidth or delay constrains, the stream request is dropped. If the dropping of the stream violates minimum bounds of NVS or NSC, then the solution is rejected. A new solution is started using another random ordering of vertices.

If we assume that both e_{BC} and e_{AC} are valid paths in Figure 4.3(h), then Figure 4.3(i) and Figure 4.3(j) generate two different subgraphs ($G''_{S_1^B}$ and $G'''_{S_1^B}$, respectively) which eventually contribute to two different global multi-stream multi-site content distribution graphs (chromosomes) as shown in Figure 4.3(k) and Figure 4.3(l), respectively. Here, we assume that subgraphs for other streams are unchanged. Therefore, by using random ordering of vertices and random selection of stream sources, we can generate different unique solutions (chromosomes). These solutions are then added into a sample pool for selection.

C. Selection based on cQoS Priority

The concept of priority-based MOOP lies in the selection process. A priority-dependent selection scheme, based on *tournament selection* [86], is used for the selection process

of chromosomes from the sample pool. As we explained earlier, each chromosome represents a multi-site multi-stream distribution graph satisfying the minimum quality of the activity requirements and bandwidth constraints. Using the subgraphs available in the chromosome, we can construct a global graph and measure the average values of each cQoS parameter (an example is shown in Section 4.6). Each chromosome is then *ranked* against all other chromosomes based on these average cQoS values in order of cQoS priorities. To allow variation (known as "chromosome diversity") in the selection process, we also consider *crowding distance* [86] of the chromosomes in the comparison. Below we first discuss how to compute the rank ρ_i and the crowding distance Γ_i for a chromosome c_i.

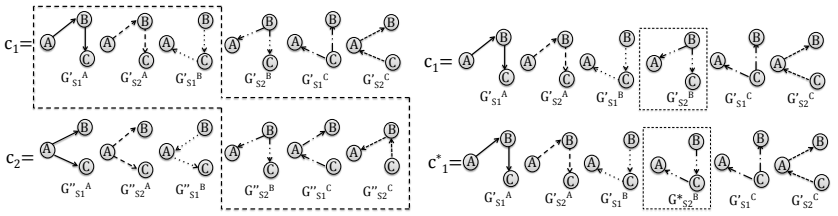
Rank: The rank is a vector containing the numerical values of the cQoS parameters and can be represented as $\rho = [|x_1|, \cdots, |x_m|]$, where $|x_i|$ is the value of cQoS parameter x_i. We assume that the cQoS parameters are prioritized as $x_1 > x_2 > x_3 > \cdots > x_m$. If we have two rank values $\rho_i = [|x_1|, \cdots, |x_m|]$ and $\rho_j = [|x'_1|, \cdots, |x'_m|]$ for chromosomes c_i and c_j, respectively, the relationships between the rank values can be represented as follows:

- $\rho_i > \rho_j$ if and only if $|x_t| > |x'_t|$ and $\forall(1 \leq q < t) \quad |x_q| = |x'_q|$.
- $\rho_i = \rho_j$ if and only if $\forall(1 \leq q \leq m), \ \ |x_q| = |x'_q|$.

Crowding Distance: The crowding distance Γ_i of chromosome c_i is a measure of the objective space around c_i, which is not occupied by any other solution in the current solution set. It helps to avoid local optima in the search space. To measure the crowding distance for chromosome c_i, we sort the population set based on their rank values and then use the following equation on the ordered set of chromosomes: $\Gamma_i = \sum_{x \in X}(|x|_{c_{i+1}} - |x|_{c_{i-1}})$, where $|x|_{c_{i+1}}$ and $|x|_{c_{i-1}}$ are the values of x for chromosomes c_{i+1} and c_{i-1}, respectively, from the sorted set. The crowding distance for the first and last chromosome in the set is set to infinity (∞).

Selection Process: Once we compute the rank value and the crowding distance, the comparison in the selection process works in three steps: (1) chromosome c_i wins over chromosome c_j (i.e., $c_i > c_j$) if $\rho_i > \rho_j$, (2) if $\rho_i = \rho_j$, then $c_i > c_j$ if $\Gamma_i > \Gamma_j$, (3) otherwise, one of them is selected randomly. Only top M_{sample} chromosomes are selected from the sample pool and stored in a *mating pool*. The mating pool contains a list of chromosomes, which are used for *crossover* and *mutation* described next.

Here we present an example of the selection process. Let consider the previous 3DTI setup for a TI conversation activity, where S_1^A, S_1^B and S_1^C are the audio streams and S_2^A, S_2^B and S_2^C are the upper body video streams. For a given chromosome, we can compute the average rate of audio streams $R_{S_{audio}}$ and average rates of video streams $R_{S_{video}}$, where $S_{audio} = \{S_1^A, S_1^B, S_1^C\}$ and $S_{video} = \{S_2^A, S_2^B, S_2^C\}$. Other cQoS parameters are average EED, average NVS and average NSC. We consider the rank with a 5-tuple $\rho = \{R_{S_{audio}}, R_{S_{video}}, EED, NVS, NSC\}$. We assume that for chromosome c_1 as shown in Figure 4.3 (k), the value of rank is ρ_1 = [$64kbps$, $2500kbps$, $300ms$, 2, 1] and for chromosome c_2 as shown in Figure 4.3 (l), the value of rank is ρ_2 = [$256kbps$, $1000kbps$, $300ms$, 2, 1]. According to the definitions of



(a) (b)

Figure 4.4: (a) An example of crossover at the subgraph of stream S_2^B. The dotted box represents the offspring chromosome c_1'. The remaining parts create another offspring c_2', (b) An example of mutation at the subgraph of stream S_2^B. $G'_{S_2^B}$ is replaced by a random graph $G^{\star}_{S_2^B}$.

the TI conversation in Section 4.2.2, the priority ordering of the cQoS is $R_{s_{audio}} > R_{s_{video}} > EED > NVS > NSC$. Therefore, based on the rank definition above, c_2 has higher rank than c_1, since the value of higher priority cQoS parameter $R_{s_{audio}}$ is higher in c_2, even though the value of lower priority cQoS parameter $R_{s_{video}}$ is lower compared to c_1.

We consider another chromosome c_3 as shown in Figure 4.3(a) that has rank value ρ_3= [$64kbps$, $2500kbps$, $300ms$, 2, 1], which is equal to ρ_1. We can sort the chromosomes in order of their rank as $\{c_2, c_1, c_3\}$ (where c_1 and c_3 are ordered randomly since they have equal rank values). Based on the above definition, the crowding distances of c_2 and c_3 are ∞. The crowding distance for c_1 is $\Gamma_1 = 192$. While performing selection between c_1 and c_3, since $\rho_1 = \rho_3$, we compare them using crowding distances. In this case, c_3 wins since $\infty > 192$. Therefore, if we are to select two chromosomes into the mating pool, we select c_2 and c_3.

D. Crossover

We perform crossover between each pair of chromosomes in the mating pool. For crossover, we adopt the concept of common node [87]. Let us consider two chromosomes for the 3DTI setup used in Section 4.5.2, $c_1 = G'_{S_1^A}|G'_{S_2^A}|G'_{S_1^B}|G'_{S_2^B}|G'_{S_1^C}|G'_{S_2^C}$ and $c_2 = G''_{S_1^A}|G''_{S_2^A}|G''_{S_1^B}|G''_{S_2^B}|G''_{S_1^C}|G''_{S_2^C}$. A crossover is performed between them if a matching subgraph is found, i.e., $G'_S = G''_S$ for any $S \in \{S_1^A, S_2^A, S_1^B, S_2^B, S_1^C, S_2^C\}$. The crossover creates two offspring chromosomes by exchanging the distribution graph at the matching point. An example is shown in Figure 4.4(a). Since, $G'_{S_2^B} = G''_{S_2^B}$, a crossover is performed at the subgraph for S_2^B. The constructed offsprings are $c_1' = G'_{S_1^A}|G'_{S_2^A}|G'_{S_1^B}|G''_{S_2^B}|G''_{S_1^C}|G''_{S_2^C}$ and $c_2' = G''_{S_1^A}|G''_{S_2^A}|G''_{S_1^B}|G'_{S_2^B}|G'_{S_1^C}|G'_{S_2^C}$.

If subgraphs are matched at multiple locations, then a random location is considered for offspring construction. If the constructed offspring chromosomes satisfy the minimum cQoS quality constraints and bandwidth constraints, then they are inserted into the sample pool.

E. Mutation

To construct a mutant chromosome from the mating pool, we randomly pick a stream and replace the subgraph of that stream with another random distribution subgraph, which satisfies the minimum quality constraints and bandwidth constraints shown in (1) and (2) of the minimum quality problem. The algorithm for building the random subgraph is the same as the initial selection process.

An example of mutation is shown in Figure 4.4(b) over the chromosome c_1 of the previous example. We randomly pick the subgraph for stream s_2^B and replace the distribution graph $G'_{S_2^C}$ using another randomly populated distribution graph $G^{\star}_{S_2^B}$. It creates a mutant chromosome $c_1^{\star}$. Note that in all stages of the genetic operation, we only consider the graphs that meet the minimum quality requirements. The mutant chromosome is then added into the sample pool.

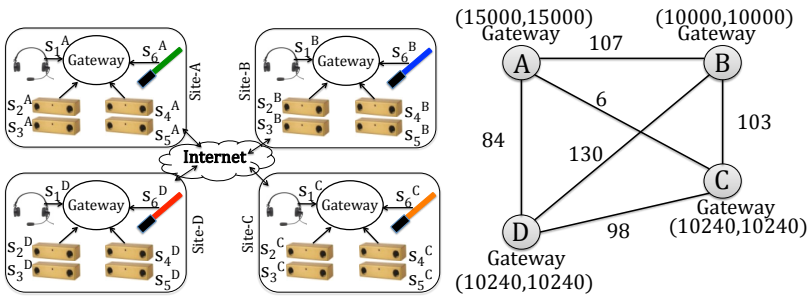
F. Optimization Procedure

The optimization process starts with the initial samples generated by ViewCast and moves towards the near optimal solution. The whole process is iterated for a fixed number of times (MG) or the consecutive L number of iterations that generates the same result. The steps in the optimization process are below.

1. Set $i \leftarrow 0$; Construct a set of initial solutions of size M_{sample} using ViewCast and add them into a sample pool.
2. Perform cQoS-priority-based selection over the chromosomes in the sample pool based on rank and crowding distance and create a mating pool of size M_{mating}.
3. Perform crossover between each pair in the mating pool and add the valid offsprings into the sample pool.
4. Perform mutation for each chromosome in the mating pool and add the valid mutant into the sample pool.
5. Select the best c from the sample pool based on rank and crowding distance. If the same c is repeated for L consecutive times, return c, else keep count for c.
6. $i++$; If ($i < MG$) , Go to step 2, else, select the best chromosome c from the sample pool and return c.

4.6 Illustrative Examples

The previous section provides a model for constructing content distribution graphs considering cQoS specifications and content demands. Here, we present two concrete examples with larger setup: one for TI conversation and the other for TI virtual lightsaber fight activities with four sites. The activity descriptions are given in section 4.2.2. All participants are using the same view orientation. Therefore, the priority ordering of the remote streams are equal for each participant. We make this assumption for the ease of explanation, however, violating this assumption does not require any changes in the algorithm. We conside the session dynamism in the next section.



(a) (b)

Figure 4.5: (a) TI application setup (showing only input devices), and (b) corresponding overlay communication graph showing end-to-end delay (along edges) and available in bound and outboud bandwidth (at vertices).

4.6.1 Input Space

3DTI Setup. The list of input devices (each device is connected to an individual PC) connected at each participating site is shown in Figure 4.5 (a). The TI conversation session uses 1 audio and 4 video streams (two upper body streams and two lower body streams) from each site. On the other hand, the virtual lightsaber session generates an additional input sensory stream connected to the lightsaber, which indicates the position of the lightsaber in the virtual space. Table 4.3 shows the list of streams generated by the application setup.

Communication Graph. The complete network communication graph between the participating gateways $V = \{A, B, C, D\}$ is shown in Figure 4.5(b). Labels on the edges define the end-to-end delay (in ms) between gateways. Each vertex is labelled with a two-tuple, (ibw_v, obw_v) $(v \in V)$, which is inbound and outbound bandwidth capacity (in kbps) of the site.

cQoS Specifications. As explained before, the cQoS specifications are defined as the cQoS priority and their minimum bounds required for an activity. Below we describe the cQoS specifications for TI conversatoin and virtual lightsaber activity.

- In a TI **conversation activity**, each participant requires at least 2 streams (audio and the highest priority upper body video) from each remote site (total of 6 remote streams), i.e., $NSC \geq 2$ and $NVS \geq 1$. The stream request for each participant and the priority of the streams are shown in the first and second rows of Table 4.3, respectively. Participants mainly focus on the audio (s_{audio}) and the highest priority upper body video (S_{ub_H}) of the remote participants (as mentioned before, the priority of the video streams can be measured using the view orientation). Therefore, to ensure a strong QoE in the conversation session, the application first needs to ensure the high quality (i.e., high bit rate) delivery of S_{audio} and S_{ub_H} before ensuring the delivery of other streams given the resource limitation. The end-to-end delay can be relaxed compared to the other collaborative activities, however it needs to be optimized before considering the

Property	TI conversation	TI virtual lightsaber
cQoS Bound	$128 \leq R_{S_{audio}} \leq 256$, $R_{S_{light}} = 0$, $2 \leq NSC \leq 5$, $1 \leq NVS \leq 4$, $EED \leq 500$, $1500 \leq R_{S_{ub\_H}} \leq 2500$, $1000 \leq R_{S_{ub\_L}}, R_{S_{lb\_H}}, R_{S_{lb\_L}} \leq 2500$	$64 \leq R_{S_{audio}} \leq 256$, $5 \leq R_{S_{light}} \leq 10$, $4 \leq NSC \leq 6$, $2 \leq NVS \leq 4$, $EED \leq 200$, $1000 \leq R_{S_{ub\_H}}, R_{S_{ub\_L}}, R_{S_{lb\_H}}, R_{S_{lb\_L}} \leq 2500$
cQoS Priority	$R_{S_{audio}} > R_{S_{ub\_H}} > EED > NVS > R_{S_{ub\_L}} > R_{S_{lb\_H}} > R_{S_{lb\_L}} > NSC > R_{S_{light}}$	$EED > R_{S_{light}} > R_{S_{ub\_H}} > R_{S_{lb\_H}} > NVS > R_{S_{ub\_L}} > R_{S_{lb\_L}} > NSC > R_{S_{audio}}$

Table 4.2: cQoS specification of TI activities (rates in kbps and delays in ms)

optimization of the quality of the lower priority upper body video streams and any lower body video streams. Another important cQoS parameter is NVS. We want to maximize the number of video streams (in the order of stream priority) that each site receives (at the minimum rate) before maximizing the rate of the low priority streams. Since, for TI conversation, we do not have any other sensory channels except audio and video, the maximization of NSC is not important.

- In a **virtual lightsaber fight**, each participant uses a lightsaber sensor (to indicate the lightsaber position in the virtual space) to virtually hit other participants and gain points to win. For a successful lightsaber session, we require both the highest priority upper body video stream (S_{ub_H}) and the highest priority lower body video stream (S_{lb_H}) to represent the full human body along with S_{audio} and the lightsaber sensory stream (say, S_{light}), i.e., $NSC \geq 4$ and $NVS \geq 2$. Also, the end-to-end delay consideration is more critical in the lightsaber game compared to the conversation activity, because, having a large end-to-end delay (EED) may impact the hitting efficiency of the participants. Therefore, before maximizing the quality of the video streams, the application first ensures the lowest possible end-to-end delay. The quality of the lightsaber sensory stream is important for detecting hitting accuracy in the virtual space. Also, increasing the number of video streams (NVS) improves the lightsaber gaming experience. Since, the minimum specification of NSC already considers the inclusion of lightsaber stream, audio stream, and the highest priority video streams, maximization of NSC is less important (note that the maximization of number of video streams is already covered by NVS). The audio quality is considered as the least important cQoS parameter for this activity.

Based on the above definitions, we give an example of cQoS quality bounds and priority orderings in Table 4.2. The cQoS parameters are: average rate of audio streams ($R_{S_{audio}}$), average rate of highest priority upper body video streams ($R_{S_{ub\_H}}$), average rate of lowest priority upper body video streams ($R_{S_{ub\_L}}$), average EED, average NVS, average rate of highest priority lower body video streams ($R_{S_{lb\_H}}$), average rate of lowest priority lower body video streams ($R_{S_{lb\_H}}$), average rate of light sensory stream ($R_{S_{light}}$), and average NSC. The averages are taken over all stream requests

Property	TI conversation	TI virtual lightsaber
Stream Type	$S_{audio} = \{S_1^A, S_1^B, S_1^C, S_1^D\}$, $S_{ub\_H} = \{S_2^A, S_2^B, S_2^C, S_2^D\}$, $S_{ub\_L} = \{S_3^A, S_3^B, S_3^C, S_3^D\}$ $S_{lb\_H} = \{S_4^A, S_4^B, S_4^C, S_4^D\}$, $S_{lb\_L} = \{S_5^A, S_5^B, S_5^C, S_5^D\}$, $S_{light} = \{S_6^A, S_6^B, S_6^C, S_6^D\}$	
Stream Priority	$S_{audio} > S_{ub\_H} > S_{ub\_L} > S_{lb\_H} > S_{lb\_L}$	$S_{light} > S_{ub_H} > S_{lb\_H} > S_{lb\_H} > S_{audio} > S_{ub\_L} > S_{lb\_L}$

Table 4.3: Application setup for TI activities

from all participating sites. The maximum bit rate of the streams are computed using the maximum media quality and the maximum media frame rate we allowed in our 3DTI setup.

4.6.2 Prioritized Evolutionary Optimization

Figure 4.6 shows an example of the iteration process in the OSM session optimization for TI conversation. Each chromosome is represented by 20 subgraphs; one for each stream (1 audio and 4 video streams from each site) in the current TI session. Since TI conversation does not use lightsaber sensory streams, no subgraph is generated for streams in S_{light}. Figure 4.6(a) and (b) show two chromosomes c_1 and c_2 constructed using ViewCast as part of the generation of initial solutions for genetic iterations. Graphs with empty edges represent the dropped streams.

During crossover, two offsprings are generated by mixing c_1 and c_2 at the common subgraph position. Since $G'_{S_1^B} = G''_{S_1^B}$, a crossover is performed at the subgraph for S_1^B. Figure 4.6(c) shows one of the offspring chromosomes generated after crossover (dotted boxes represent the subgraphs coming from c_1 and solid boxes represent the subgraphs coming from c_2). A mutation is then performed at the subgraph $G''_{S_3^B}$ over the generated offspring. The mutant chromosome $c^\star$ is shown in Figure 4.6(d). Both the offspring and mutant chromosomes are added into the sample pool for the next iteration. Similar process can be shown for TI virtual lightsaber activity; however skipped due to the space limitation.

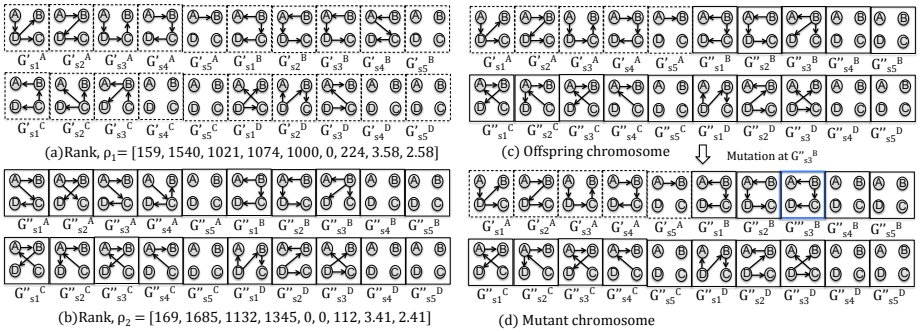


Figure 4.6: Optimization steps in TI conversation: (a) and (b) chromosome c_1 and chromosome c_2, respectively created using ViewCast, (c) offspring c'_1 created using crossover between c_1 and c_2 at subgraph for S_1^B, and (d) mutant $c^\star$ created by replacing the subgraph for S_3^B in c'_1 .

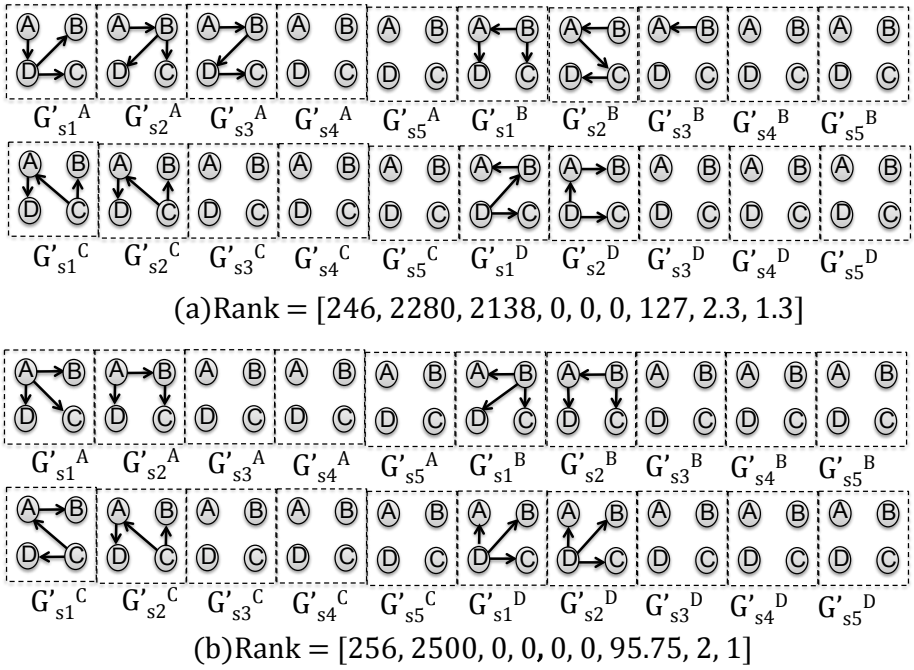


Figure 4.7: TI Conversation(a) after 2 iterations, and (b) optimized solution.

Here, the rank is a 9-tuple vector ρ = [$R_{S_{audio}}$, $R_{S_{ub\_H}}$, $R_{S_{ub\_L}}$, $R_{S_{lb\_H}}$, $R_{S_{lb\_L}}$, $R_{S_{light}}$, EED, NSC, NVS]. Though the computation of EED, NSC and NVS are straightforward from a given distribution graph, the allocation of rates for individual stream is tricky since the distribution graph is constructed considering the minimum bit rates of the streams. To allocate stream rates, after a distribution graph (or chromosome) is constructed with minimum rates, for each path in the graph, we allocate rates to the streams (associated to the path) in order of their priority (as shown in Table 4.3) subject to the bandwidth availability. For example, for c_1 in Figure 4.6(a), the directed edge e_{AD} is shared by four streams S_1^A, S_3^A, S_2^B and S_3^B. In the conversation activity, $S_1^A > S_2^B > S_3^A > S_3^B$. So, first we assign the minimum rates for each stream along the edge. The remaining bandwidth is then first assigned to S_1^A, if surplus is available after the maximum rate allocation to S_1^A, the remaining amount is assigned to S_2^B and so on.

Using the above approach, the ranks of c_1 and c_2 are computed as ρ_1 and ρ_2 and shown in Figure 4.6(a) and 4.6(b), respectively. Since, there is no light sensory stream in the TI conversation activity, the rate associated to it is zero. As c_2 assigns higher rate to the audio stream, due to the priority based comparison in the selection step, c_2 wins over c_1.

We finally run the optimization process iteratively using GA. Figure 8(a) shows the solution after 2 iterations and Figure 8(b) shows the optimized solution after 20 iterations. The rank values are also given below the figures. Even the intermediate result (Figure 8(a)) in GA provides higher rank compared to the initial solutions c_1 and c_2

(shown in Figure 4.6(a) and Figure 4.6(b), respectively). The optimized solution in Figure 8(b) assigns the the average rate of the audio stream to $256kbps$, and the average rate of the highest priority upper body video stream to $2500kbps$, which are the maximum possible in both cases. It also lowers the EED to $95.75ms$. However, the lower body video streams are completely dropped (i.e., low NVS and NSC). This is acceptable since according to our activity definition, having only audio stream and upper body video streams from each site are enough for a successful TI conversation activity.

Figure 9 shows the optimized distribution graph for TI lightsaber activity with 24 streams (each site has 8 streams). It first optimizes the end-to-end delay (to $88ms$) and then the rate of lightsaber sensory stream (to $7.5kbps$). Streams S_{ub_H} and S_{ub_L} maintain the minimum quality ($\geq 1000kbps$). The rate of the audio streams are given a lower priority in optimization. If we compare the optimized rank values between TI conversation and TI lightsaber activities, it is evident that both the distribution graphs are optimized considering the requirements of the respective activities.

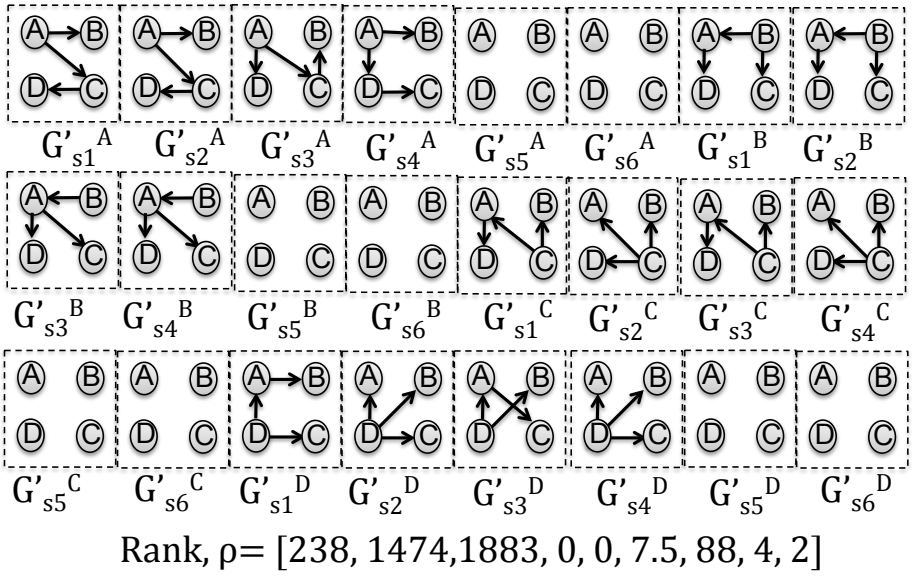


Figure 4.8: Optimized graph for TI lightsaber activity.

4.7 Handling Session Dynamism

During session initiation, OSM constructs a multi-stream content dissemination graph considering the stream priority (based on participants' views), resource constraints and 3DTI activity. However, during the activity phase, participants may change their views and underlying bandwidth resources may reduce. Any of these session updates needs to re-optimize the QoS allocation for multi-stream content distribution according to Section 4.5. This optimization process can be expensive (e.g. few seconds), which may violate the 3DTI interactivity requirement. Therefore, in this section, we propose

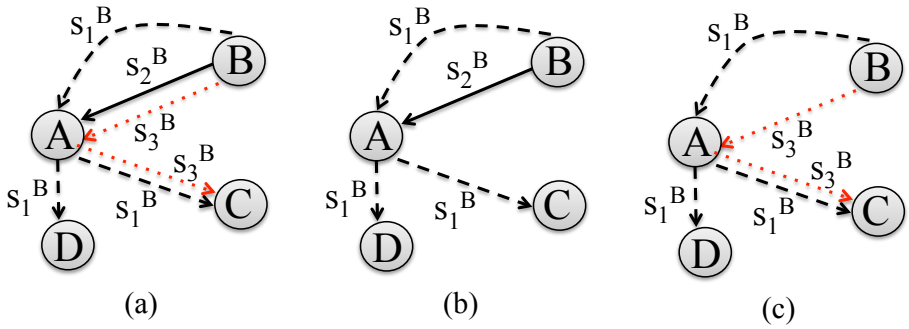


Figure 4.9: Example of instantaneous session update, a) before Site-A requests a session update (bandwidth drop), b) after S_2^B is dropped at Site-A, and c) after S_3^B is dropped at Site-A during instantaneous session update.

a two-step session adaptation process that reacts promptly when a session update is required.

In the two-step session adaptation, when a session update is triggered, a temporary session modification is performed by the GSC instantaneously to address the current changes occurred in the participants' views and/or the network resources. The computation of cQoS optimization (based on genetic algorithm) is performed in the background. Once the GSC gets the optimization result, the session routing tables of the running session are updated at the participating gateways. We call this approach a two-step session adaptation since the updates of the session routing tables at the gateways are done in two steps for each session update. Details of the adaptation process are given below.

4.7.1 Instantaneous Session Adaptation

When a 3DTI site requests a session update (e.g. view change or resource update), the instantaneous session adaptation process running inside the **QoS optimizer and topology constructor** module (in Section 5.4) at the GSC adjusts the multi-stream distribution graph by dropping, adding or modifying rates of one or more streams at the requesting site. Other cQoS parameters as well as cQoS priorities are not considered to keep the instantaneous update fast and simple. However, the selection of streams at the requesting site is not trivial since the dropping or modifying rates of a stream at the requesting site may impact other receiving sites along the forwarding path in the multicast-based dissemination. To minimize the impact, GSC computes *impact factor* for each stream possible adjustment (e.g., drop or rate reduction) that can be made at the requesting site, and picks the one with the least impact factor.

To understand how the stream selection impacts the overall 3DTI performance during the instantaneous session adaptation, we present a simple example in Figure 4.9. The inbound bandwidth at Site-A is 5.5Mbps. Based on the participant's view at Site-A, the stream priorities are $Pr^A_{S^B_1} = 3$, $Pr^A_{S^B_2} = 2$ and $Pr^A_{S^B_3} = 1$. However, the stream priorities at Site-C are $Pr^C_{S^B_3} = 3$, $Pr^C_{S^B_1} = 2$ and $Pr^C_{S^B_2} = 1$ based on the partici-

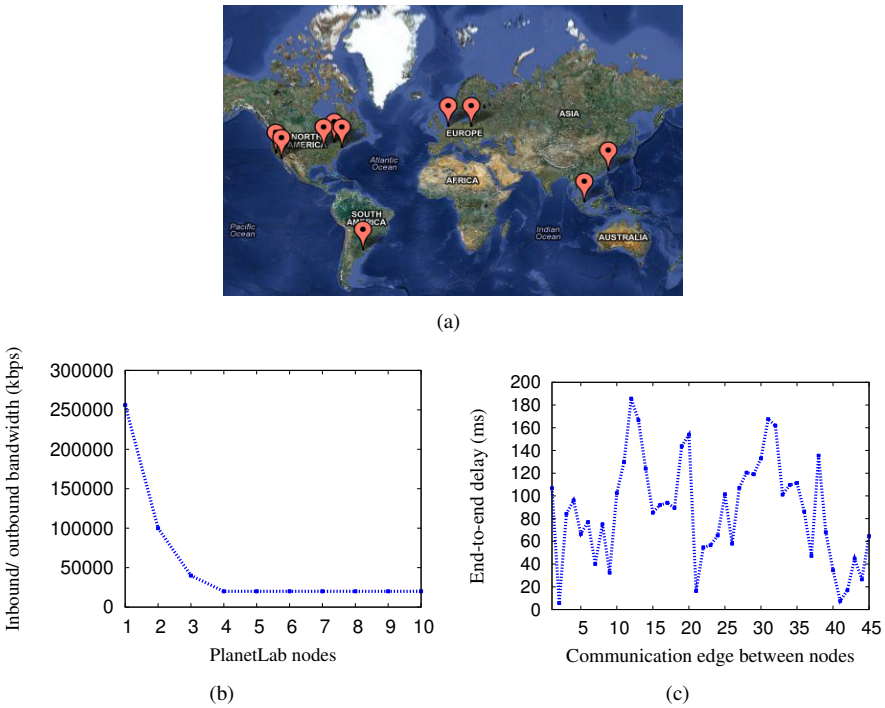
pant's view at Site-C. Without loss of generality, we assume that streams are transmitted to Site-A at their minimum acceptable rates ($R^A_{S^B_1} = 2.5$Mbps, $R^A_{S^B_2} = 1.5$Mbps, $R^A_{S^B_3} = 1.5$Mpbs) for the intended activity. If the inbound bandwidth at Site-A drops to 4Mbps, the session adaptation needs to drop one incoming stream. Considering the stream priority at Site-A, if we drop the lowest priority stream S^B_3 at Site-A (Figure 4.9(b)), it impacts Site-C since Site-C is receiving S^B_3 via Site-A, which is the highest priority stream for Site-C. On the other hand, S^B_2 at Site-A is not forwarded to any other sites and therefore, dropping of S^B_2 at Site-A is more meaningful (Figure 4.9(c)) considering the global performance of the system. To formalize this stream selection process during instantaneous session adaptation, we quantify the impact of different stream adjustment using the impact factor metric. The impact factor (IF) is defined as follows:

Impact Factor: The impact factor of stream s at site-X is computed by the priority of s at site-X and at other successive sites along the multicast links from Site-X, and based on the current and adjusted rates. If L defines the set of successive sites along the multicast path, $R^l_s(new)$ is the adjusted rate (0 for dropped stream) and $R^l_s(old)$ is the old rate of s toward Site-l, the impact factor for modifying s at Site-X is computed as follows: $IF_s = \sum_{l \in L}(1 - \frac{R^l_s(new)}{R^l_s(old)}) \times Pr^l_s$, where Pr^l_s is the priority of s at Site-l.

Based on the above definition, we can compute the impact factor for dropping stream S^B_1, S^B_2 and S^B_3 at Site-A in Figure 4.9. Stream S^B_1 at Site-A is forwarded to Site-C. Hence L=$\{A, C\}$. The impact factor for dropping S^B_1 at Site-A: $IF_{S^B_1}$=8 (where $Pr^D_{S^B_1}$=3). Likewise, we can compute $IF_{S^B_2}$ (=2) and $IF_{S^B_3}$ (=4). Since based on the values, $IF_{S^B_2} < IF_{S^B_3} < IF_{S^B_1}$, we drop S^B_2 for instantaneous adaptation. In the above example, we consider that a stream needs to be dropped due to the bandwidth scarcity. However, it is possible that instead of dropping streams, we can adjust streaming rates. But the goal of instantaneous adaptation is to keep the modification latency short and hence, we only consider the rate adaptation over minimum number of streams rather than dropping different rates of multiple streams for an optimal performance.

4.7.2 Optimized Session Adaptation

The final phase in the session adaptation is the evolutionary session optimization. The optimization step considers the overlay after instantaneous update as one of the initial solutions in the adaptation process since it represents known close-to-optimial solution in the optimal search space. Once the optimized session is computed, the GSC installs the session routing table to the participating gateways to update the content dissemination graph. Though during the instantaneous session adaptation, some participants may receive the lower priority streams (e.g., S^B_2 is dropped rather than S^B_3 in Figure 4.9), the global session performance is kept higher. Finally, the optimization step ensures the QoE maximization based on the underlying activity.



(a) (b) (c)

Figure 4.10: (a) Geographical distribution of PlanetLab nodes, (b) Inbound/ outbound bandwidth distribution of 10 nodes, and (c) end-to-end delay between the nodes (for 10 nodes, there are 45 edges).

4.8 Performance Analysis

4.8.1 Simulation Setup

We generate a mesh-based complete graph of an application-level overlay network, where the connectivity (delay and bandwidth) between the vertices follows a distribution collected from PlanetLab [39]. The total number of vertices, denoted as the *session size*, ranges from 3 to 10. The geographical distribution of the PlanetLab nodes used as vertices in our evaluation is shown in Figure 4.10 (a). We take vertices distributed widely over the world. Using a running service [40] from the PlanetLab, we collect outbound bandwidth information of each node (the lowest outbound we measure is $20Mbps$). We assume that no other slices on the same node are competing for the bandwidth. For simplicity, we set the inbound bandwidth same as the outbound bandwidth measured. The end-to-end delay between the vertices are measured by `ping` inside the nodes. The distribution of bandwidth as well as the end-to-end delay between the vertices are shown in Figure 4.10(b) and (c), respectively.

In the experiment, each vertex has 16 video streams, which are evenly distributed in the 360^0 space; 8 covering upper body and 8 covering lower body of the participants, one audio stream and one light sensory stream. For a view request to a vertex, at most 8 of its original streams are selected for an optimal coverage of upper and lower body of 180^0 space along with the audio and sensory streams. The optimization process (as

in 4.5 is implemented using java.

We consider two types of TI activities: conversation activity and virtual lightsaber fight activity as described in Section 4.6. The priority of the cQoS parameters for these TI activities and their minimum and maximum bounds are same as described in Table 4.2. However, the number of streams and vertices are variable in our simulation. To have a successful conversation session, each vertex should receive at least one (the highest priority) upper body video stream and the audio stream from each other vertex. On the other hand, to have a successful virtual fight session, each vertex should receive at least two video streams (the highest priority upper body video stream and the highest priority lower body video stream), the audio stream and the light sensory streams from each remote vertex.

4.8.2 Performance of QoS Allocation

To measure the performance of genetic session optimization technique, we compare the rank of the distribution graph constructed by OSM with the rank of the graphs constructed by two other techniques: 1) ViewCast [4], and 2) stream-based random multicast [17]. In random multicast, the priority of the streams are not considered and multicast trees are constructed randomly; however bounded by the end-to-end delay. We vary the number of nodes (selected from Figure 4.10(a)) in our evaluation and each experiment was run one hundred times.

Genetic optimization is continued until 3 consecutive searches provide the same (best) result. The sample pool size in GA is set to 10 and the mating pool size is 5. The ranks of the solutions are computed using the same technique shown in Section 4.6. For both activities, we only show the values of $R_{S_{audio}}$, $R_{S_{ub\_H}}$, $R_{S_{lb\_H}}$, EED, NVS, and NSC in Figure 4.11(a)-(f). In case of TI conversation, the OSM provides the best audio (in Figure 4.11(a)) and upper body video (in Figure 4.11(b)) bit rate (hence, better quality) with the sacrifice of end-to-end delay (in Figure 4.11(d)), NSC (in Figure 4.11(e)) and NVS (in Figure 4.11(f)). Compared to the ViewCast, the maximum improvement in the audio bit rate is about 50% (for 7 vertices in Figure 4.11(a)) and compared to the random distribution topology, the maximum improvement is about 25% (for 6 vertices in Figure 4.11(a)). Since, for a successful conversation, lower body of the video streams are not required, in OSM, the rates of them are very low (in Figure 4.11(c)); however, ViewCast and random distribution topology assign higher bit rate to them.

On the other hand, for TI virtual lightsaber activity, the end-to-end delay (in Figure 4.11(d)) is optimized along with the quality of the lightsaber sensory streams (not shown) by OSM. The quality of the video streams is balanced between the highest priority upper body video stream (Figure 4.11(b)) and the highest priority lower body video stream (Figure 4.11(c)) to construct a full body human image rather than trying to solely optimize the quality of only one portion of the body streams. Though the audio quality is poor (in Figure 4.11(a)); it still maintains the minimum quality requirement for TI virtual lightsaber activity as defined in Table 4.2.

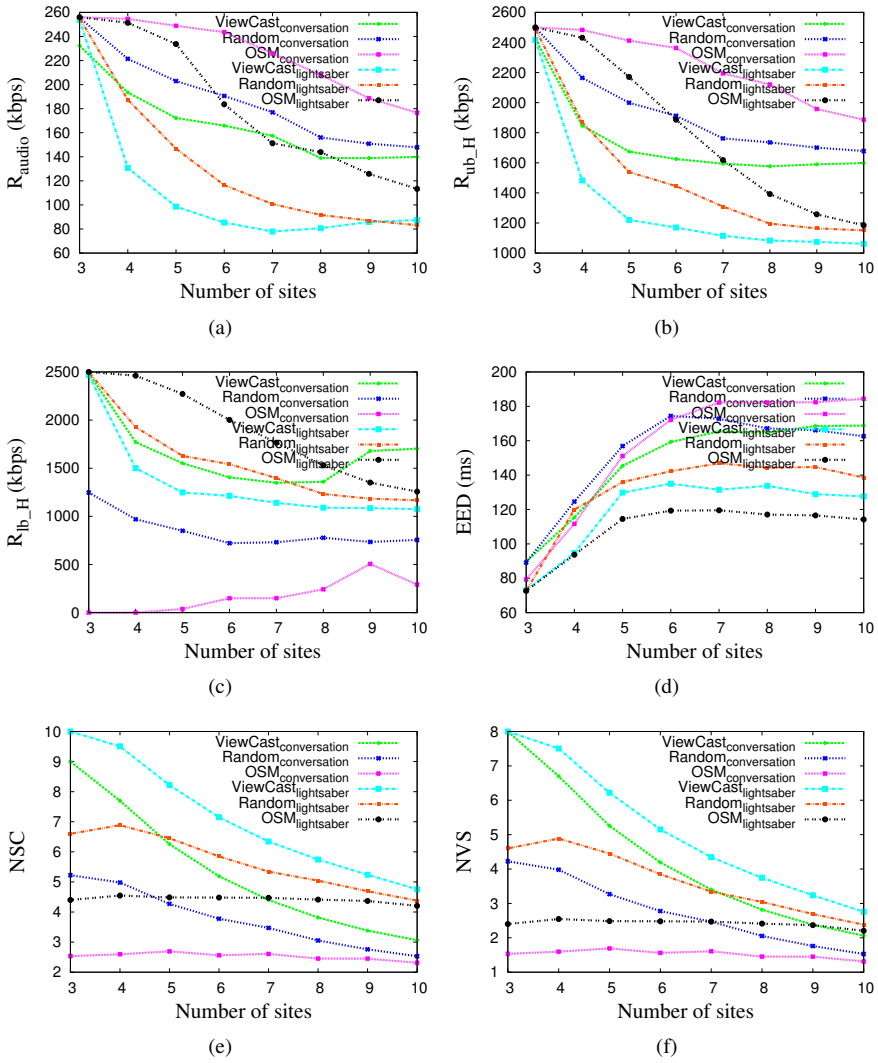


Figure 4.11: (a) Average rate of audio quality, (b) average rate of highest priority upper body video stream, (c) average rate of highest priority lower body video stream, (d) average end-to-end delay, (e) average number of streams received from each remote site, and (f) average number of video streams received from each remote site.

Figure 4.11(e) and Figure 4.11(f) show that the values of NVS and NSC are always higher in ViewCast for both TI conversation and TI virtual lightsaber activities. This is because ViewCast is originally designed to maximize the number of video streams in a TI session. However, both ViewCast and random distribution topology create higher end-to-end delay for TI virtual lightsaber and lower audio bit rate for TI conversation, compared to the OSM, since none of them consider prioritized cQoS based on the on-going activity requirements. Therefore, according to the definitions of the activities, OSM provides higher user satisfactions in both activities compared to the ViewCast and random multi-site and multi-graph distribution topology.

4.8.3 Performance of Interactivity

To measure the performance of 3DTI interactivity using OSM, first, we show how OSM impacts the session adaptation latency, and then we formalize how the viewing quality of the participants are interfered by the session update.

A. Response Time

A significant portion of latency in OSM session adaptation process comes from the iterations performed in the genetic optimization: selection, crossover and mutation (Section 4.5). However, the instantaneous session adaptation masks the optimization latency. Below we show the latency of both steps individually. Here, we only use TI conversational activity, where each site can request maximum of 9 streams from each remote site.

Prioritized Optimization: The computational latency increases with the increase of the number of iteration to perform in the optimization process. Moreover, the genetic operations at each iteration becomes more expensive to execute when the overlay graph becomes larger. We plot the execution latency of genetic optimization over different iterations and different number of graph size (by varying the number of participating sites) in Figure 4.12(a). Though it takes 4sec to perform 20 iterations for 10 sites, the required number of iterations to generate an optimal solution is less than 20.

To understand how many iterations are required to generate a near optimal solution, we plot the iteration count by varying the number of participating sites as well as the number of streams to request from each remote site in Figure 4.12(b). We consider a solution is near optimal when consecutive 5 iterations generate the same best solution for content distribution graph. From the Figure 4.12(b), the number of iterations required for an optimal solution is higher for smaller number of participating sites as well as for smaller number of stream request. The reason is that having a small demand on the content distribution process creates a large solution space to search for an optimal solution. Because of this conflicting nature, the session optimization latency is less than 2 seconds as shown in Figure 4.12(c).

Instantaneous Session Modification: The latency for instantaneous update is very small since the computation is only responsible for finding the impact factors for all possible stream adjustments at the update requesting site. Moreover, the multicast path length is bounded due to the delay bound. Therefore, the latency does not vary much with the increase of number of sites. As Figure 4.12(c) shows, the instantaneous update can be performed within 300ms after a session update is requested, which is in the same order as ViewCast and Random multicast-based adaptations.

B. QoS Interruption

In addition to the response time of session adaption, we need to ensure that the adaptation process does not interfere QoS of the running session. To measure the perceived cQoS over the session run-time, we compute normalized values of the cQoS parame-

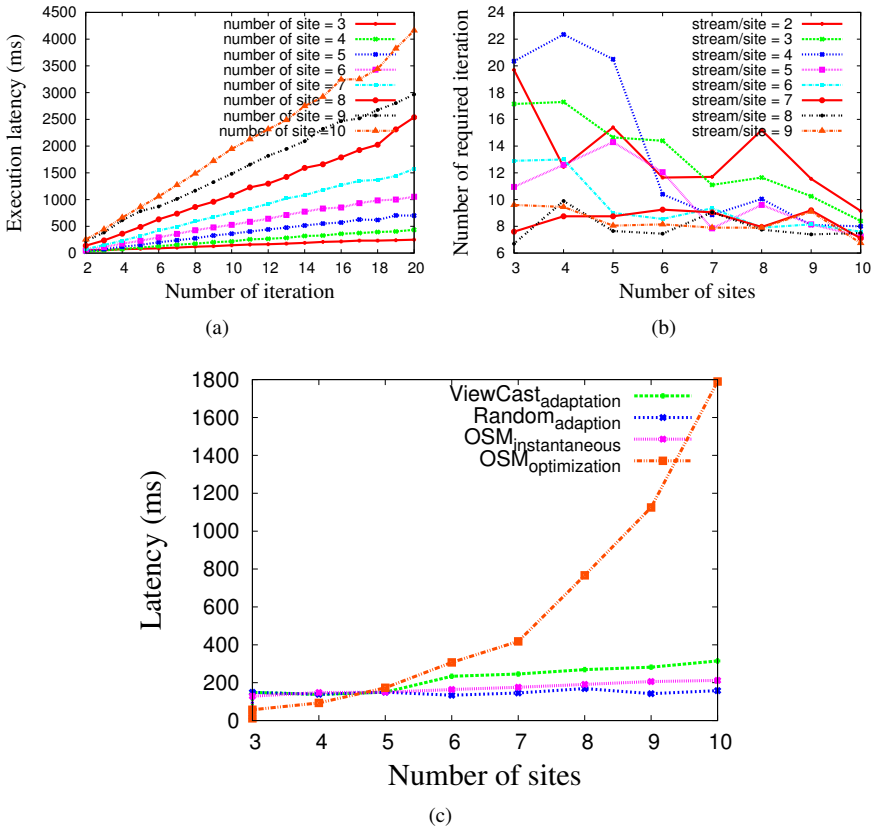


Figure 4.12: (a) Latency of session optimization for different number of iterations in OSM. (b) average number of iterations required to achieve near optimal solution in OSM, and (c) latency of session initiation and session optimization.

ters. The normalized cQoS or NQ is defined as follows: $NQ = 1/m \sum_{i=1}^{m}(x_i/max_{x_i})$ for m cQoS parameters. Note that we cannot use NQ as the rank value (ρ) since it does not consider cQoS priority in the equation. To use NQ as a session quality metirc, in this section, we only consider view-change-based session adaptation. During a view change, some old streams are dropped and some new streams are restored. Dropping of old streams lowers the NQ value, and restoration of new streams increases the NQ value provided that other streams and their rates are non-variant, (which we ensure during the session run-time).

We measure NQ of the multi-stream dissemination graph at each 500ms interval for 2.5 minutes of a 3DTI session among 10 participants. Results are shown in Figure 4.13(a) and 4.13(b). Vertical lines indicate the view changes made by the participants during the session run-time. In Figure 4.13(a), no instantaneous session adaptation is used. When a site requests a view change, it drops the old streams and requests for the new streams. Since the streaming of the new streams are stored only after the optimization is done, we see a dip (lower value) in the NQ during the view change. On the other hand, Figure 4.13(b) shows that using instantaneous session adaptation, the degradation of NQ is limited. It is also interesting to note that when multiple users

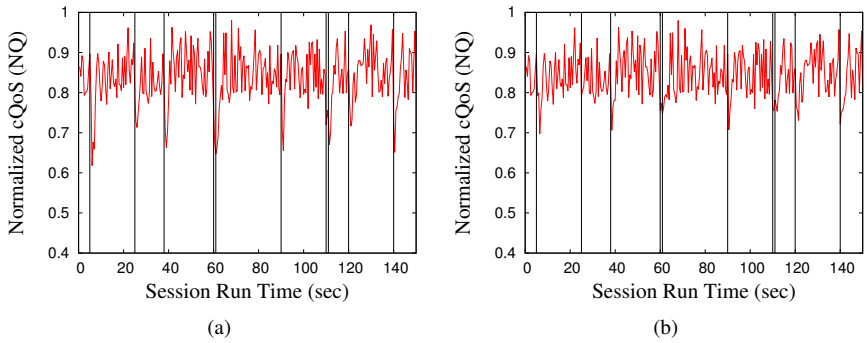


Figure 4.13: Normalized cQoS of 3DTI session in (a) one step: genetic optimization, and (b) two steps: instantaneous adaptation and genetic optimization.

change views at the same time (where the vertical lines are close), the cQoS interruption becomes high. However, the variance is usually very small.

4.8.4 CPU and Memory Overhead of OSM

Since OSM performs session optimization inside a centralized session controller, the overhead penalties of CPU and memory are of important concerns. In our experimentation, even with the 10 participating sites each requesting 9 streams from all other remote sites, the memory overhead is less than $100MB$, which is very low compared to off-the-self PC configurations. To measure the CPU overhead, we compute the CPU utilization of the optimization process by varying the number of participating sites and the number of input streams per site. The system configuration of GSC is: 2.8 GHz Intel Core $i7$ running Mac OS X (v. 10.7.4). Our results show that the CPU overhead is around $20\% - 30\%$.

4.8.5 Subjective Evaluation

We use ITU standard in conducting the subjective experiment. We recruit 23 adult participants (age ranging between 18 and 45). The sample consists of 7 females (30%) and 16 males (70%). We consider two activities: video conferencing and virtual lightsaber between two sites using two cameras (upper and lower body) and one audio stream. We vary number of video streams, video bit rate, audio bit rate, and end-to-end delays (in the form of audio-visual skew) during the session run-time. We artificially limit network bandwidth and impose delays in the network during session run-time so that the cQoS parameters require an adaptation. We record videos of two sessions: session A and session B. Session A is running OSM based adaptation, however, session B drops lower-body video stream during the lightsaber activity, and reduces bit rate of video streams during the video conferencing activity.

Each subjective participant is asked to compare two videos, one from Session A (OSM-based) and the other from Session B (non-OSM-based) in a 5-point scale $A>>B$ (session A is much better than session B), $A>B$ (session A is slightly better than ses-

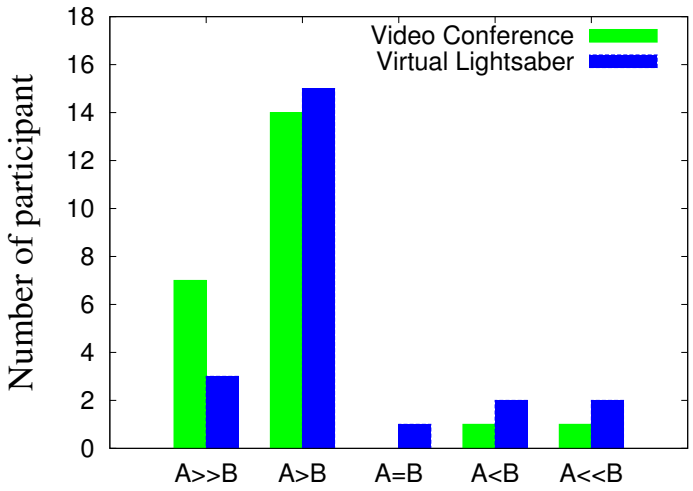


Figure 4.14: OSM subjective comparison between Session A and Session B.

sion B), $A{=}B$ (session A is same as session B), $A{<}B$ (session B is slightly better than session A), and $A{<<}B$ (session B is much better than session A) for each 3DTI activity. We record 1.24 minutes of conversation in video conferencing, and 52 seconds of gaming for virtual lightsaber activity for both sessions. The subjective result is shown in Figure 4.14(c) for both activities. In the video conferencing activity, about 91.3% participants prefer the OSM based 3DTI session (i.e., 21 out of 23 participants consider OSM-based session A much or slightly better than non-OSM-based session B). On the other hand, for the lightsaber activity, about 79% (18 out of 23) prefer OSM. Only 2 participants in the video conferencing and 4 participants in virtual lightsaber gaming prefer non-OSM-based 3DTI session (i.e., session B). On the other hand, no participant in the video conferencing and only 1 participant in the virtual lightsaber activity consider session A and session B the same. OSM clearly improves the QoE of the participants since the session is constructed considering the participant's preference in the running activity.

4.9 Conclusion

This chapter presents an evolutionary optimization technique for 3DTI session management considering the QoS specifications (minimum quality bounds and QoS priorities) defined by the 3DTI activity and content demands. We use three heuristics in GA to improve the 3DTI session optimization performance: 1) ViewCast-based generation of initial samples (solutions), 2) prioritized QoS-based sample selection, and 3) crowding distance-based sample variation. We show that our heuristics-based GA converges very fast and provides about 50% improvement in the allocation of desired QoS parameters. We also show that using a two-step heuristic, we can mask the latency of genetic optimization during session adaptation.

5 Network Layer Support in Immersive Session

5.1 Introduction

Building an optimized session by allocating QoS parameters (introduced in the previous chapter) works well to construct a user-expectation-aware 3DTI session. However, we have not taken any network layer support to explore further improvement in end-to-end data transmission in terms of latency and resource consumption. With the advancement of Software Defined Networking (SDN) research over the past few years, network components have become accessible and controllable from application layer [24]. Applications can use this flexibility to offload certain data plane functionalities to the network layer for the purpose of improving data plane complexity and efficiency. We have seen the leverage of SDN (e.g., OpenFlow [88]) in different applications such as selecting servers for load balancing [69], managing consistency in network migration [71], and finding peers for video streaming [73]. The natural question for us is "can we use OpenFlow network components to reduce data plane complexities and improve multi-stream flow management in immersive 3DTI". To address this question, we propose *OpenSession, a session-network cross-layer management control protocol* for managing multi-stream flows in immersive 3DTI.

OpenSession introduces a split forwarding architecture at the application data plane. It *partially* offloads stream multicast functionality of the application to the SDN switches. If multiple participants request the same streams (defined by the overlay constructed in Chapter 4), the source participant sends only single copies of the local streams to the SDN network switch, which handles multi-stream forwarding to all remote destinations on-behalf-of the application. The splitting of data plane reduces processing load at the application, and network bandwidth usage at the network. Offloading multi-stream forwarding to the SDN network component also improves end-to-end delay in multicast-based streaming because forwarding of streams is done from the network layer of the SDN switches instead of from the application layer of the end-hosts at each multicast hop.

However, splitting the data plane introduces several challenges. First, the application layer multi-stream semantics are lost in the network layer data plane. Without keeping this semantics, the SDN switches cannot forward streams according to the overlay topology. To solve the semantic mapping problem, OpenSession develops a *packet differentiation* mechanism that assists the switches to differentiate data packets across multiple streams. Second, SDN provides a fixed interface (OpenFlow) to configure the network layer data plane. To work with this standard interface, OpenSession develops

a *mapping algorithm* to map the multi-stream overlay topology from the application layer to the network layer. Third, 3DTI requires frequent updates on the data plane configuration due to frequent view changes. To allow fast and seamless view changes, the updates on the data plane should be performed consistently across all participants so that the view changes do not introduce any interference on streaming. To maintain a global consistency while updating the distributed data planes, OpenSession develops a *coordinated route update protocol*.

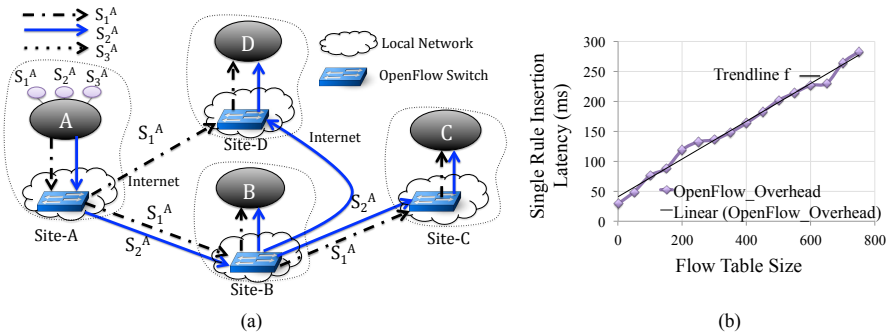
We implement OpenSession using OpenFlow switches [88] and Floodlight controller [89], where the application layer overlay are constructed using OSM as shown in Chapter 4. To prove the benefits of OpenSession, we run experiments with real 3DTI application setup among four distributed participants (within small geographical region). To show the impact of larger scale on Internet, we also run distributed 3DTI sessions using PlanetLab [39] nodes. Our results show that OpenSession improves multi-stream 3D streaming performance. It reduces end-to-end delay in proportion to the round-trip time (RTT) of local networks (e.g., a campus network where the application host resides) at each multicast hop. It also improves bandwidth load within the local network and processing load inside the application host in proportion to the number of streams in a 3DTI session.

Though we focus on interactive 3DTI throughout this paper, OpenSession does not provide any design limitation that bounds other multi-party tele-conferencing applications to take advantage of it. Our key contributions are:

- We design and implement OpenSession, a session-network cross-layer management algorithm for multi-stream 3DTI using SDN support (Section 5.4 and Section 5.5).
- We demonstrate (Section 5.7) that OpenSession improves 3DTI interactivity in proportion to RTT of the local network, and reduces bandwidth load within the local network and processing load inside the application host in proportion to the number of streams.
- We show (Section 5.7 and Section 5.8) that OpenSession makes the data plane resilient against frequent view changes, route updates and host failures. It also introduces very low control overhead in terms of latency and processing.

5.2 Software Defined Networking

In Software Defined Networking, a logically centralized controller manages how the network switches forward packets in layer-2 and layer-3. The controller exchanges control messages with the switches using a standard protocol, such as OpenFlow [66]. Therefore, in SDN, switches are often called OpenFlow switches. We use the terms "SDN" and "OpenFlow" interchangeably in this chapter. The control plane installs *forwarding rules* to forward packets in the switch data plane using a *flow table* with *match entry* and *action field*. Incoming data packets are matched against the match



(a) (b)

Figure 5.1: (a) Example of a 3DTI session among 4 sites, (b) OpenFlow layer-3 rule insertion latency for different sizes of flow table.

entry, and corresponding actions (e.g., forwarding, header modification) in the action field are performed on the matched packets. OpenFlow actions also allow modification of packet headers on the data forwarding plane.

5.3 Motivation and Challenges

Before presenting the technical details of OpenSession architecture, in this section, first, we demonstrate potential benefits of using OpenFlow switches to offload application layer stream forwarding functionalities in 3DTI (Section 5.3.1). Later (Section 5.3.2), we point out key challenges that OpenSession should address.

5.3.1 Benefits

Consider the scenario shown in Figure 5.1(a). Site-A generates three streams (for simplicity, we only show streams generating from a single site) S_1^A, S_2^A and S_3^A. S_1^A and S_2^A are distributed to Site-B, Site-C, and Site-D during a running 3DTI session. S_3^A is dropped due to inbound bandwidth limitation. The local network shown in the figure can be a campus network, a department network, or a home network. In OpenSession, streams are forwarded from the network switches, connected within the local network.

Local bandwidth: OpenSession introduces a significant reduction of 3DTI bandwidth within the local network of participating sites. For example, in Figure 5.1(a), forwarding of streams S_1^A and S_2^A to Site-C and Site-D is done from the network switch of Site-B rather than from its application gateway. Therefore, the gateway of Site-B does not forward streams S_1^A and S_2^A from the application host, which saves the forwarding bandwidth of S_1^A and S_2^A within the local network of Site-B. It evetually saves the last mile bandwidth to and from the application host. Bandwidth saving also happens at the streaming source. Though Site-A needs to send stream S_1^A to both Site-B and Site-D, only one copy traverses the local network to the local OpenFlow switch, which performs stream replication and forwarding.

End-to-end delay: Since forwarding of stream from Site-B is done from the network layer switch (in Figure 5.1(a)), the end-to-end delays (EEDs) of streams S_2^A from Site-

A to Site-C and Site-D are reduced by the round-trip time (RTT) of 3D frames in the local network (d_{net}) and through the protocol stack of the gateway machine (d_{app}) at Site-B. To measure the order of improvement in EED, we monitored our campus network for 3 days from January 15, 2013 to January 17, 2013. We used four gateway hosts (over wired and wireless links) at different locations within the campus, and measured point-to-point RTT among them for exchanging 3D frames. The average EED is about 4ms to 5ms, which is a significant latency in real-time communication considering a single multicast hop.

Processing load: Though Figure 5.1(a) considers only forwarding of two streams, multi-stream 3DTI constructs a much more complex distribution graph. We have experienced that with the increase of number of sites and the number of streams per site, the forwarding load (CPU load) of 3DTI application layer gateway increases linearly. Since, in OpenSession, the network layer is responsible for stream replication and forwarding, the application layer is not impacted by the increase of number of subscribers (i.e., participants) of a stream.

System resiliency: Application hosts fail and crash more frequently than underlying network components. As OpenSession separates multicast forwarding from the application layer, the failure of an application gateway or any crash of application due to a software bug at one site does not interfere the real-time collaborative experience among other participating sites. Failure of B does not affect S_1^A to Site-C in Figure 5.1(a). However, we skip this benefit from the current scope and leave it for future research.

5.3.2 Challenges

Although attractive performance gains are seen from the examples above, many design challenges remain to realize the expected benefits.

Per-stream packet differentiation: To forward streams from the local OpenFlow switches according to multi-stream overlay distribution graphs, OpenSession must ensure application-aware differentiation of network packets at the switches. For example, stream S_2^A (in Figure 5.1(a)) from Site-B is forwarded to both Site-C and Site-D, whereas stream S_1^A is only forwarded to Site-C. Therefore, the OpenFlow switch at Site-B should be able to differentiate network packets of S_1^A and S_2^A. OpenFlow standard provides a fixed set of fields to match against incoming packets [88]. However, traditional source address, destination address, source port and destination port based packet differentiation fails since multiple streams among the sites use the same network layer flow semantics. Therefore, we need to develop a mechanism so that the network packets from different application streams can be differentiated at the switches. To use the benefit of OpenFlow-based split data plane architecture, we need to override certain fields in the packet header so that packets from different streams can be identified uniquely and proper forwarding actions (defined by the multi-stream overlay topology graph) can be taken on them at the switches. we need to ensure two properties in *packet differentiation*: 1) overridden fields are not modified anywhere in the transmission path,

otherwise we lose the semantic information at the remote sites, and 2) the design does not require any changes in network components or OpenFlow protocol.

Overlay to flow table mapping: Next, we need to ensure that flow table rules are consistently installed at all the participating OpenFlow switches to follow the stream *forwarding actions* according to the overlay. The forwarding actions include the transmission of data packets defined by the multi-stream overlay topology defined by OSM. However, blind mapping of overlay to flow tables fails since it may interrupt other running flows in current or concurrent 3DTI sessions (e.g., via overriding or superseding available flow table entries). We need to ensure that the flow table size is optimized in the mapping process.

To understand the impact of switch flow table size on 3DTI session, we have conducted an isolated experiment with IBM G8264 OpenFlow switch [90].We measure a single rule insertion latency for different number of pre-installed rules into the switch. The result is shown in Figure 5.1(b). Wildcarded layer-3 rules are stored in a 'linear' table in the TCAM space of the switch [91]. As TCAM is expensive and small, every wildcarded layer-3 rule insertion performs an internal optimization of flow tables [92], which represents significant part of the rule insertion latency. This optimization latency increases with the increase of flow table size, which impacts session initiation and session update latency. Therefore, we need to ensure two properties while mapping the multi-stream overlay topology to switch flow tables: 1) modification of flow table does not interrupt on-going forwarding actions, and 2) size of flow table is as small as possible.

Seamless view change: View change introduces a new human behavior in the multi-stream 3DTI environment. If the participant at Site-B updates a view, which requests streams S_2^A and S_3^A instead of S_1^A and S_2^A, then stream S_1^A is no longer required by Site-B. However, if Site-A stops transmission of S_1^A to Site-B, the streaming path of S_1^A to Site-C is impacted and therefore, should be reconfigured. Sites who are impacted by remote view changes are called *victim sites*. Streaming paths for all impacted 3DTI streams to victim sites should be updated before the actual view change can take place. Here, Site-C is the victim site due to the view change by Site-B. To ensure seamless streaming to Site-C due to this view change, the update of streaming path for S_1^A to Site-C should not interfere the immersive experience at Site-C in terms of packet loss.

Therefore, to ensure seamless view changes, the creation of new topology and the removal of old topology for streams should be performed concurrently. For example, if we update the path $A \rightsquigarrow D \rightsquigarrow C$ in Figure 5.1(a) before removing the path $A \rightsquigarrow B \rightsquigarrow C$ for S_1^A, then Site-C may get transiently overloaded with redundant S_1^A. It is undesirable as 3D streams consume large bandwidth. Likewise, if we remove the path $A \rightsquigarrow B \rightsquigarrow C$ first and then update rules to create $A \rightsquigarrow D \rightsquigarrow C$, it introduces temporal disconnectivity of S_1^A to Site-C. With frequent view changes, the viewing experience of the victim sites can be very poor. What is required is a concurrent update of distributed flow tables that ensures two properties: no transient 1) bandwidth overloading, and 2) network disconnectivity.

5.4 OpenSession Framework

OpenSession uses the same three-layer session management hierarchy as shown in Figure 1.6. Their functionalities are described below with an example:

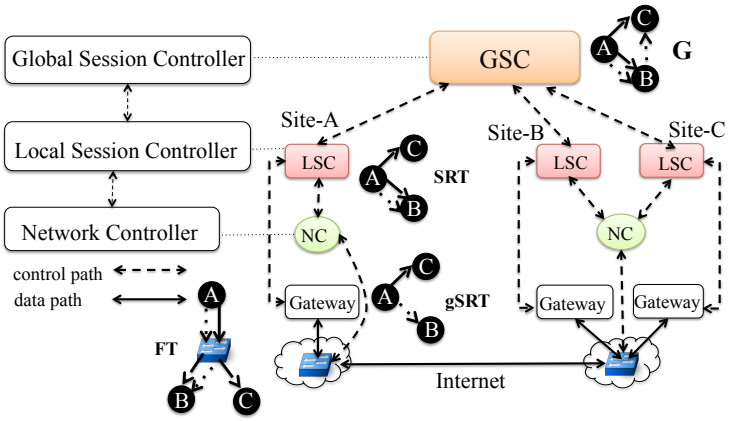


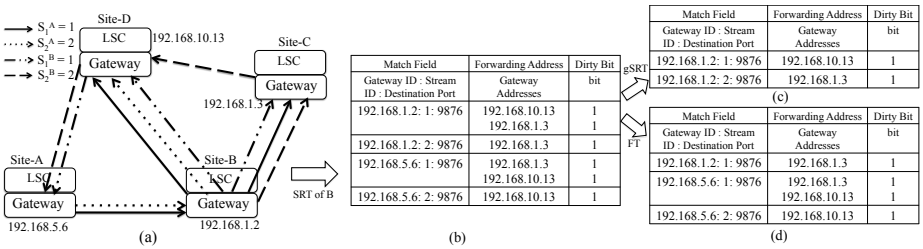
Figure 5.2: OpenSession framework.

Global session controller: GSC represents a global control plane. We use the same GSC as we shown in Chapter 4. The outputs of the GSC are SRTs for participating sites, which define match field (SRT_{match}) and forwarding actions (SRT_{action}) based on the optimal overlay. We do not consider the rate allocation in our current scope. In Figure 5.2, we show an example graph corresponding to the SRT of site-A for a multi-stream overlay G computed at the GSC (streams from Site-A are only shown).

Local session controller: LSC is a lightweight (in terms of processing overhead) session layer as in OSM. The main responsibility of LSC is to update session routing table (called gSRT) at the local gateway (i.e., at the application data plane) using the SRT received from GSC (Section 5.5.1). Gateways use gSRT for the data plane forwarding of local streams. gSRT ensures that single copies of local streams are delivered out of the gateway. Figure 5.2 shows an example of gSRT at the gateway of Site-A. LSC forwards SRT to local SWC to map it to the switch flow table (FT). LSC is also responsible for monitoring local devices and resources, and sends them to GSC.

Network controller: NC is the new controller layer, that we use solely for OpenSession. It directly communicates with the OpenFlow switches connected in the local network to install forwarding actions defined by SRT (Section 5.5.1). An example of forwarding actions in FT related to the SRT and gSRT of the previous example is shown in Figure 5.2. NC is also responsible for maintaining consistency and size optimization in FT. We run NC as a module of the OpenFlow controller.

Data plane: The data plane is separated from the management control plane and divided between the application gateway and the OpenFlow switch. The gateways schedule local streams to multicast remotely based on the connectivity graph defined by gSRT. The OpenFlow switches multicast local and remote streams based on the forwarding actions defined in FT.



Match Field	Forwarding Address	Dirty Bit
Gateway ID : Stream ID : Destination Port	Gateway Addresses	bit
192.168.1.2: 1: 9876	192.168.10.13 192.168.1.3	1 1
192.168.1.2: 2: 9876	192.168.1.3	1
192.168.5.6: 1: 9876	192.168.1.3 192.168.10.13	1 1
192.168.5.6: 2: 9876	192.168.10.13	1

(b)

Match Field	Forwarding Address	Dirty Bit
Gateway ID : Stream ID : Destination Port	Gateway Addresses	bit
192.168.1.2: 1: 9876	192.168.10.13	1
192.168.1.2: 2: 9876	192.168.1.3	1

(c)

Match Field	Forwarding Address	Dirty Bit
Gateway ID : Stream ID : Destination Port	Gateway Addresses	bit
192.168.1.2: 1: 9876	192.168.1.3	1
192.168.5.6: 1: 9876	192.168.1.3 192.168.10.13	1 1
192.168.5.6: 2: 9876	192.168.10.13	1

(d)

Figure 5.3: (a) Multi-stream overlay topology for S_1^A, S_2^A, S_1^B, and S_2^B, (b) the corresponding SRT of Site-B, (c) SRT entries that map to gSRT at Site-B (application data plane), and (d) SRT entries that map to OpenFlow FT serving at Site-B (network data plane). 9876 is UDP data communication port.

5.5 OpenSession Protocol and Functionality

A 3DTI Session is initiated by sending a session initiation request to the GSC, which contains a list of sites, a list of local devices (streams) at each site, and a list of desired views for all participants. Designing an efficient content distribution overlay that can combine different notions of quality (e.g., startup delay, frame rate) and user experience is an open challenge that is outside the current scope. Here we focus on OpenSession protocols for: (1) mappings of multi-stream content distribution topology to application and network layer data planes, and (2) handling of seamless changes in the distribution topology during session run-time.

5.5.1 Multi-stream Overlay Mapping

In this section, we will discuss, how the multi-stream overlay is mapped into SRT (Section 5.5.1), gSRT (Section 5.5.1) and finally to OpenFlow FT (Section 5.5.1).

A. Overlay Topology to SRT Mapping

Once the multi-stream overlay topology is computed using OSM, it is mapped to the SRT for each participating site. The mapping of overlay to SRT aims to associate forwarding actions with each stream. The entries of SRT are already shown in Section 4.4.2. However, we do not consider SRT$_{rate}$ in our current OpenSession implementation. We leave it for future research. An example of SRT computation from a multi-stream overlay is shown in Figure 5.3. For clarity, we only show two streams originating from Site-A and Site-B. The overlay topology computed by GSC is shown in Figure 5.3(a). Figure 5.3(b) shows the corresponding SRT of Site-B.

B. SRT to gSRT Mapping

The purpose of mapping SRT to gSRT is to define the forwarding responsibility to the application layer data plane at the gateway. Each gateway sends only single copies of the local streams towards the local OpenFlow switch, though multiple sites may request the same streams. Streaming towards multiple sites are then done from the

OpenFlow switch via address modification (destination IP and MAC addresses) in the packet header. Therefore, the mapping from SRT to gSRT is done only for local streams to ensure that the gateway sends only one copy. In Figure 5.3(b), Site-B sends stream S_1^B to both Site-C and Site-D. However, the LSC only maps the streaming path to Site-D in gSRT at Site-B (application data plane). The streaming to Site-C is done from the network layer data plane. Figure 5.3(c) shows the entries of SRT that are mapped to gSRT.

C. SRT to FT Mapping

Finally, the mapping of SRT to FT is performed by NC in three steps below:

Prune SRT entry: We only map dirty SRT entries. Moreover, we prune the SRT entries (associated to the local streams), which are already assigned to the application layer data plane (i.e., to gSRT). For example, in Figure 5.3(b), the SRT entry to forward S_1^B (with Match Field $192.168.1.2 : 1 : 9876$) to Site-D is pruned and not mapped to FT at Site-B. However, the SRT entry to forward S_1^B to Site-C is not pruned.

Update match field: After pruning SRT, we map SRT match field (SRT_{match}) to FT match field (FT_{match}). The OpenFlow protocol allows direct mapping for source IP address (gw) and destination port (P_{dst}). However, the notion of application layer stream ID (S_{id}) is lost in the network layer data plane. To keep this semantic information in the network layer, we override the higher 5 bits of *IP ToS (Type of Service)* field. Overriding of IP ToS bits is very common in Internet applications as they are less frequently used in Internet [93]. When stream data is sent by the gateway, the 8bit IP ToS field of the UDP packets is set based on 5 bits of stream ID with zeros in lower 3 bits (i.e., we do not modify *ECN* bits used for congestion control). For example, if $S_1^A = 1$, the corresponding ToS field in the IP data packet is 00001000. Therefore, $\text{SRT}_{match} = [192.168.5.6.1 : 1 : 9876]$ maps to $\text{FT}_{match} = [192.168.5.6.1 : 00001000 : 9876]$.

Update action list: In addition to forwarding the incoming packets to the intended destination (based on packet header), the OpenFlow switches need to forward the same packets to other sites (based on $\text{SRT}_{actions}$) by replacing the destination IP and destination MAC addresses. If the destination IP is in the remote subnet, OpenSession uses MAC address of the gateway-router. Also, the physical port for each forwarding action is assigned based on the destination IP. In summary, in addition to forwarding the packets based on their original headers, we add three actions in FT_{action} field for each forwarding IP x in SRT_{action}: 1) set destination IP to x, 2) set destination MAC address to MAC_x, and 3) forward the packets to port Port_x. Here, Port_x is the physical port number at the switch that forwards data packets to destination x, and MAC_x is the MAC address required to forward data packets to destination x.

5.5.2 Seamless Session Adaptation

Session adaption (adding or dropping streams) is triggered when participants change views, or bandwidth resources at the participating sites change. In this section, we

only consider view changes; other causes will naturally follow. The result of the view change is route update for one or more streams.

A. Seamless Route Update Problem

As we show in Section 5.3.2, a route update may cause a) **overloading** when redundant streams are transmitted to any participating site), and b) **disconnectivity** (when streaming of one or more streams becomes temporarily unavailable). To understand the phenomena clearly, we show a route update example with six participating sites in Figure 5.4(a)-(d) for both overloading and disconnectivity. With frequent view changes, interactive streaming performance at the victim sites due to transient overloading and disconnectivity becomes very poor.

B. Route Update Problem Formulation

The route update problem can be mapped to a *constraint-based graph transformation* problem. Initially, we will consider only single stream. Later (in Section 5.5.2) we will extend our solution for multiple streams. Let us consider an existing overlay graph (G^S_{old}) and a new overlay graph (G^S_{new}) computed at GSC after a view change request. We represent each graph by a connectivity matrix M^S, where $M^S[i][j]$=1 if Site-i forwards stream S to Site-j, otherwise $M^S[i][j]$=0 (note that for the rest of this section we consider A as 0, B as 1 and so on while indexing the connectivity matrix). The transformation of graphs can be formulated by their connectivity matrices as follows:

Graph Transformation Problem: Transform M^S_{old} (connectivity matrix of G^S_{old}) to M^S_{new} (connectivity matrix of G^S_{new}) for each stream S, where$M^S_{old} \neq M^S_{new}$, so that the intermediate states of the connectivity matrix (M^S_{int}) in the transformation process do not violate following two constraints (N= number of sites): 1) $\sum_i M^S_{int}[i][j] \leq 1$, $\forall_{0 \leq j \leq N-1}$, and 2) $M^S_{int}[i][j] \neq 0$, if $M^S_{old}[i][j]$=$M^S_{new}[i][j]$=1 $\forall_{0 \leq i,j \leq N-1}$. Constraint (1) ensures that there is no transient overloading of S, and (2) ensures that there is no transient disconnectivity for S to any site.

C. Route Update Solution Overview

To solve seamless route update in OpenSession, we consider an additional packet header field (η) in the FT match entry. When we add new rules (for the purpose of transforming M^S_{old} to M^S_{new}) with a new value in η, the new rules are not matched against the incoming streams right away, because incoming streams do not contain the new value of η in their packet header. In Figure 5.5, we provide an example of how consideration of an additional match entry (η) in FT can solve the route update problem for S^A_1 as shown in Figure 5.4. Below we describe.

Figure 5.5(a) shows the initial FT rules in Site-A before the route update. Since Site-D does not forward S^A_1, there is no FT entry at Site-D for stream S^A_1. While updating the route from Figure 5.4(a) to Figure 5.4(d), we first add a new FT rule at Site-D (in Figure 5.5(b)) with match field entry [A, S^A_1, P_{dst}, η'] (where η' is the new value of

η) and an action to forward the matched stream to Site-C. Even though Site-D receives S_1^A from Site-A, the data packets are not forwarded to Site-C as incoming packets do not contain η' and are not matched with the new rule (i.e., no violation of constraints (1)).

To initiate the matching with new entries in FT (at Site-D), the OpenFlow switch at the source site (at Site-A) needs to update η with the new value η' in the outgoing packet header while streaming (S_1^A). We replace the FT rule at Site-A that forwards S_1^A to Site-B by a new FT rule with two actions: 1) forward S_1^A to Site-D, and 2) set η with η' in the outgoing packet header. When this rule is inserted at Site-A, the data packets of S_1^A towards Site-D contain η' in the packet header and are redirected to Site-C from Site-D. It also terminates the streaming to Site-B at the same time (Figure 5.5(c)). We can control the instant when we want to switch streaming paths concurrently for a stream by modifying FT rules at the streaming source (i.e., concurrently start streaming from Site-D to Site-C and stop streaming to Site-B and Site-C). Thus, OpenSession ensures that the victim sites are not impacted due to the view change (or any session adaptation) triggered by remote sites unless the updated path creates a large jitter, which depends on route computation algorithms.

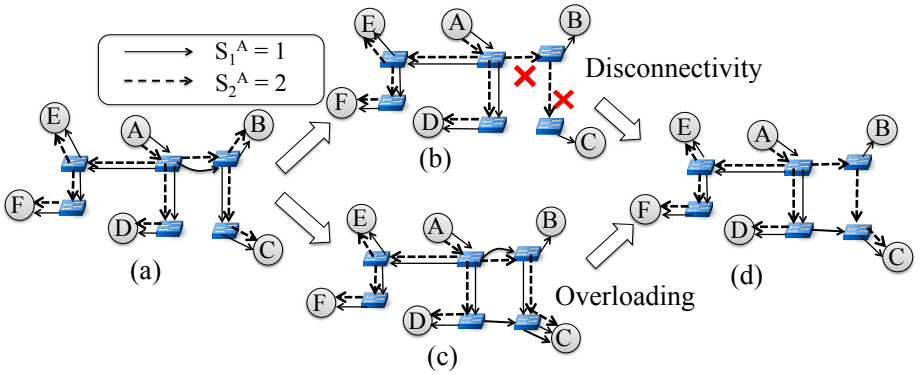


Figure 5.4: Problem of route update: (a) old overlay graph, (b) disconnectivity at Site-C, (c) overloading to Site-C, and (d) new overlay graph.

Among other layer-3 and layer-4 fields, we consider source port (P_{src}) in the UDP header as η, because changing UDP source port for the outgoing packets does not violate any routing constraints. P_{src} is monotonically increased with every route update and reset to 1024 (0 to 1023 are specific purpose ports) when all running 3DTI sessions terminate. Note that gateways do not modify P_{src} in the packet header, rather the OpenFlow switch performs the replacement on the outgoing packets of the local streams.

If we update P_{src} in the FTs only at the sites impacted by a route update, soon there will be different values of P_{src} in the FTs across all sites with the same source IP, UDP destination port and IP ToS bits, which may create inconsistency in future FT update. Therefore, when a route update happens for stream S, all distributed FT rules that forward S are updated and start matching η' in the P_{src} match field.

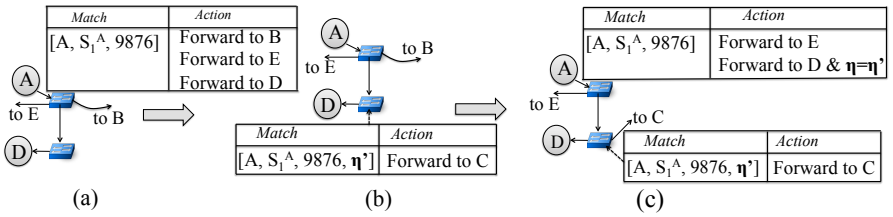


Figure 5.5: (a)-(c) Sequence of FT updates for a seamless route update of S_1^A for the case shown in Figure 5.4. 9876 is the data communication port.

Delete obsolete rules: When incoming streams are matched against the new rules, old rules becomes obsolete and are deleted automatically using the OpenFlow-defined timeout primitive. For a FT entry, if there are no matching packets for a continuous interval defined by `idle_timeout`, the rule will be deleted from the switch. We specify 30sec `idle_timeout`. OpenFlow also uses a `hard_timeout`, which removes the rule from FT after the timeout even matching flows exist. OpenSession uses infinite `hard_timeout`.

D. OpenSession Route Update Algorithm

Summarizing the above discussion, 1) we need to specify a new η each time a route update happens, and 2) FT entries in the remote sites need to be updated first before updating the FT entries at the source site (Figure 5.5(a)-(c)). Below we formally present the distributed route update algorithm at different controller levels.

Route update at GSC: GSC is responsible for initiating the route update. Depending on M_{new}^S and M_{old}^S, GSC classifies Site-i into four states (a site can be in multiple states):

1. **No-route** Site-i not forwarding S to any site before and after the route update ($\forall_j\ M_{old}^S[i][j]{=}0 \wedge M_{new}^S[i][j]{=}0$),
2. **Drop-route** Site-i drops forwarding S to any site after the route update ($\exists_j\ M_{old}^S[i][j]{=}1 \wedge M_{new}^S[i][j]{=}0$),
3. **Add-route** Site-i starts receiving S from any site after the route update ($\exists_j\ M_{old}^S[j][i]{=}0 \wedge M_{new}^S[j][i]{=}1$), and
4. **Keep-route** Site-i keeps receiving S from same site after the route update ($\exists_j\ M_{old}^S[j][i]{=}1 \wedge M_{new}^S[j][i]{=}1$).

If a site is in no-route state, no updates in SRT and FT are required and no messages are sent to the site's LSC. Likewise, if a site is only in drop-route state, no SRT and FT updates are required. Obsolete entries (no match for η') are removed by the timeout. If a site is in keep-route state, though there is no update required in SRT, an update in FT is still required to assign $P_{src}{=}\eta'$ in the FT match entry (Section 5.5.2). A route update request message Ω_{FT} is sent to the site's LSC. If a site is in add-route state, we need to update both FT and SRT to reflect the changes in the forwarding paths at

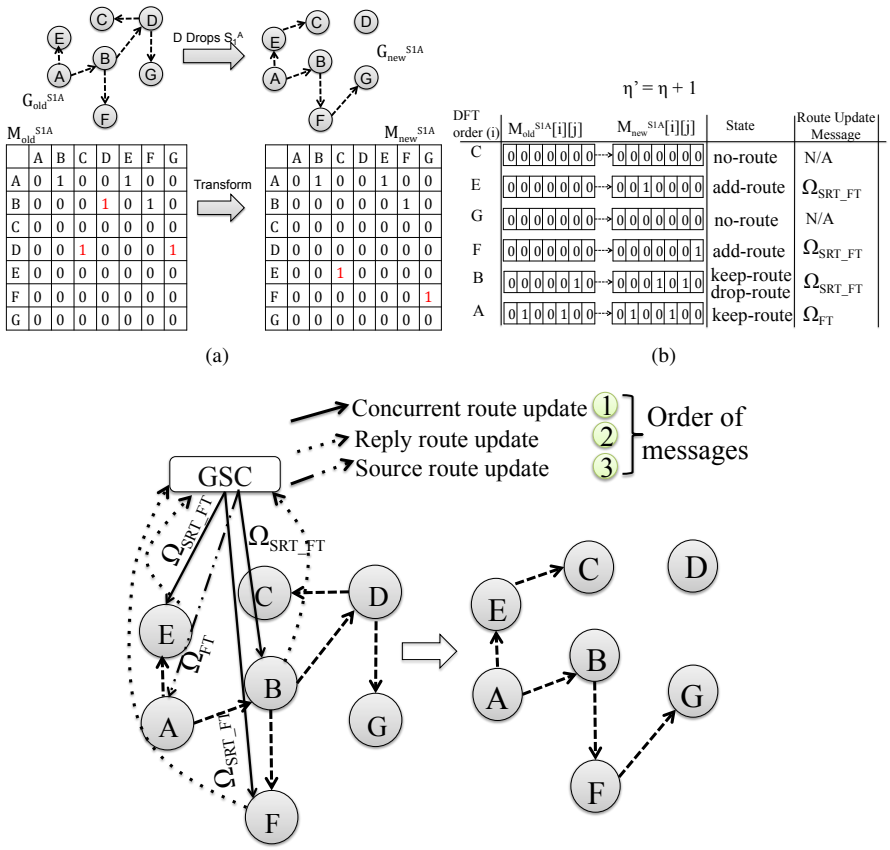


Figure 5.6: Example of OpenSession route update process for a single stream: (a) route update due to Site-D drops stream S_1^A, (b) route update messages and computed state of the sites due to the route update, and (c) route update protocol showing the order of route update messages.

Site-i. In this case, GSC sends a route update message Ω_{SRT_FT} to the site's LSC. If a site is in multiple states, the route update messages corresponding to the states are sent cumulatively to the site's LSC. Messages Ω_{FT} and Ω_{SRT_FT} are sent concurrently to all sites except at the source site of S. After all sites update their tables (SRT and FT), GSC sends a route update request message to the source site. For the source site, the selection of route update message is also made based on its state. However, even the source site is at drop-route state (i.e., streaming to one or more remote sites is dropped), we still need to send Ω_{SRT} message to update gSRT. Each route update message contains new SRT (SRT_{new}) with all dirty bits set to 1 (since we need to update P_{src} for all entries), and η'. To ensure the ordering of route update requests, GSC constructs a *depth first traversal (DFT)* of sites from M_{new}^S and sends route update requests to all sites concurrently except the last site in the order, which is the source site. Once route update requests are replied by all sites, then a route update message is sent to the source site to perform an atomic switch from M_{old}^S to M_{new}^S.

Route update at LSC: When Ω_{FT} or Ω_{SRT_FT} is received at LSC for remote streams,

it simply forwards the message to NC. However, for the local streams, LSC updates gSRT using SRT_{new} using the algorithm discussed in Section 5.5.1 and then forwards the message to NC.

Route update at NC: When NC receives route update message Ω_{FT} or Ω_{SRT_FT}, it maps the rules from SRT_{new} to FT with P_{src}=η'. Process for handling route update messages at the source site is the same, except P_{src} is not considered in the FT match entry field for S, and an additional action is added in the FT action list to replace P_{src} with η' in the outgoing packets of S. We modify our mapping algorithm in Section 5.5.1 to incorporate P_{src}, however we skip it for brevity. To ensure that the rule has been added to the FT entry, a *barrier message* is sent to the switch [88]. Reply of the barrier message confirms that the previous command for rule insertion is completed successfully. Once FT entries are updated successfully, a confirmation is sent to the GSC. Figure 5.6 shows an example of OpenSession seamless route update process for the update of stream S_1^A.

E. Multi-stream Route Update

During a view change, if routes of multiple streams are required to be updated in the new overlay, OpenSession performs source-based route updates as shown in Section 5.5.2 for all modified streaming paths in parallel provided that the route updates are *independent* among the streams. The route update messages (in Figure 5.6(d)) are sent cumulatively for all streams towards the participating sites.

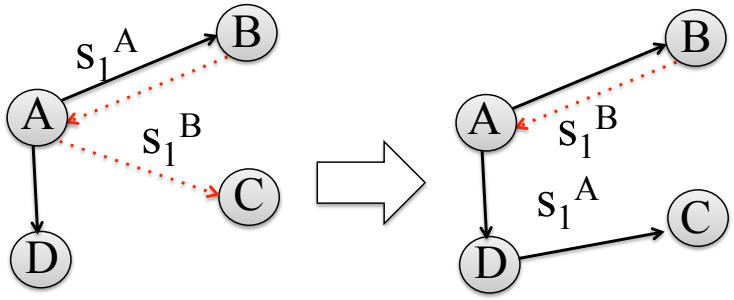


Figure 5.7: Example of dependent multi-stream route update.

However, the route updates can be *dependent*. For example, if Site-C adds receiving of S_1^A and drops receiving of S_1^B in the new overlay during session adaptation process as shown in Figure 5.7, the route updates for S_1^A and S_1^B become dependent. It is challenging to guarantee seamless session adaptation when route-update dependencies exist among the streams generated from different source sites. The source-based route update cannot guarantee atomic update of routes for S_1^A and S_1^B at Site-C as the streams are originating from separate sources. If the route updates for S_1^A and S_1^B are performed concurrently, at any point in time during the route update process we may get both S_1^A and S_1^B streaming towards Site-C . Therefore, we cannot guarantee bandwidth bound at Site-C if dependent route updates are performed concurrently.

To ensure bandwidth guarantee, we need to sequence the dependent route updates to minimize the interference in streaming and to improve seamless viewing experience.

Sequence Ordering Problem: Given N route updates (RU_S) for stream $S \in \{S_1^A, S_2^A, \cdots, S_1^B, \cdots, S_1^C, \cdots\}$, our goal is to find a sequence of route updates to transform G_{old} to G_{new}. However, a sequence of route updates to guarantee seamless view change may not exist, e.g., let us consider the case that $RU_{S_1^A}$ depends on $RU_{S_1^B}$, $RU_{S_1^B}$ depends on $RU_{S_1^C}$ and $RU_{S_1^C}$ depends on $RU_{S_1^A}$. In this case, it is not possible to find any sequence that guarantees bandwidth availability. Therefore, we develop an algorithm for finding a sequence of route updates that minimizes the number of dependency violation (or violation of bandwidth guarantee) during the overlay transformation.

Sequencing of Dependent Route Updates: To solve the route update process for multi-stream 3D tele-immersion, we uses the following steps:

- In the first step, we identify the dependent and independent route updates. Given G_{old} and G_{new}, we consider two route updates as dependent if two conditions are met: 1) streams are originated from different sources, and 2) the same site drops one stream and adds the other stream in reception, or drops one stream and adds the other stream in forwarding. The cases are shown in Figure 5.8.

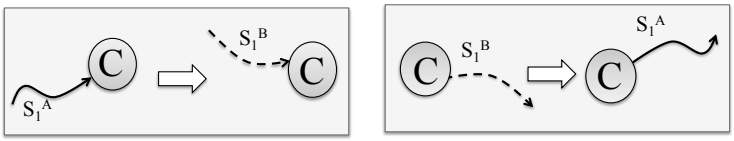


Figure 5.8: Example of dependent multi-stream route update.

- Independent route updates are performed concurrently since they do not violate the bandwidth guarantee.

- Next, we compute a *score* for each dependent route update set, where a set is constructed by "union" of dependent route update pairs with common stream. For example, if $\{RU_{S_1^A}, RU_{S_1^B}\}$ and $\{RU_{S_1^A}, RU_{S_1^C}\}$ are dependent pairs, the corresponding route update set contains $\{RU_{S_1^A}, RU_{S_1^B}, RU_{S_1^C}\}$. The score for each route-update is equal to the number of dependency violation it may create if that route update is performed performed over the current overlay. For each deponent route update set, we pick the route update with the lowest score. The selected route updates are performed concurrently. For the remaining non-empty sets, we again compute the score over the current topology and iteratively update the overlay with the route update of minimum score value.

Sufficient Condition: As we discussed before that the sequence ordering cannot guarantee the seamless view change if route-update dependencies exist. Here we present the sufficient condition in new overlay construction that can ensure seamless route updates and bandwidth guarantee. The sufficient constraint is: (**RUC**) if a site drops (adds) forwarding (receiving) of S_x^i, it does not add (drop) forwarding (receiving) of

S_y^j in the same route update process for any x and y where $i \neq j$. Sites violating RUC may not experience seamless adaptation. Note that view changes do not violate RUC since during a view change i=j (i.e., streams come from the same site). Also, when lower priority streams are dropped due to network congestion, the overlay modification does not usually violate RUC. In our evaluation for the performance of OpenSession due to view change, we consider RUC constraint in the new multi-stream topology computation.

5.6 Deployment Consideration

In the previous section, we assume different deployment supports to concentrate more on the algorithmic part. Here, we clarify the implementation and deployment issues before moving into the evaluation.

Where to place OpenFlow switch? In OpenSession, each 3DTI site is associated with an OpenFlow switch available within the local network. The placement of the switch can be very diverse, however it requires that any data traffic to and from the local gateway to remote gateways must traverse via the local OpenFlow switch associated to the local site.

What if P_{src} value is overflowed? For each view change or session update, GSC increments P_{src} to keep global consistency in forwarding rules and to optimize flow table size at the OpenFlow switches. 32bit P_{src} limits the number of session updates allowed. To allow more session updates, when P_{src} is close to overflow, we reset it to 1024.

How to define MAC in packet forwarding? While mapping the overlay to the flow table (in Section 5.5.1), OpenSession specifies destination MAC address to be replaced for each multicast-based forwarding action. To identify the destination MAC address, NC uses the ARP table from the OpenFlow switch. Forwarding to local IP addresses is done by using the corresponding MAC addresses from the ARP table. For remote IP addresses, we use the MAC address of the gateway-router.

How to find switch port for forwarding? Using ARP table of the switch, it is possible to map the physical port number to the destination IP address [89]. For remote forwarding, we push the packet via the same input port where the packets arrive to send it back to the gateway router.

5.7 Evaluation with Real 3DTI Setup

5.7.1 Evaluation Setup

We setup 4 3DTI sites (in Figure 5.9) in different types of networks. Site-A is in a *home network*, where the OpenFlow switch is placed next to the gateway. Site-B is in a *campus network*, where the OpenFlow switch is placed within the campus network. Site-C is in a *company network*, where the OpenFlow switch is placed closer to the

company's network edge. Site-D is in a *department network*, where OpenFlow switch is placed closer to the department's network edge.

Each site contains 8 video streams capturing participants 3D stereoscopic image from 45^0 apart and constructs a mesh-based color-plus-depth 3D image. Instead of using physical cameras, we read streams from stored files, which already contain the 3D video of the participants for 1 hour session. Streams are read at the original frame rate. Average bandwidth of each 3D stream is about 1.5 to 2.1Mbps. Each application node (site) hosts a gateway to multiplex transmission of local streams. We place LSC on the same gateway host and implement NC as a module of Floodlight [89] controller running on a separate host at each site. All control commands use TCP and data traffic uses UDP. We use OpenFlow software switches with 10GBps port capacity.

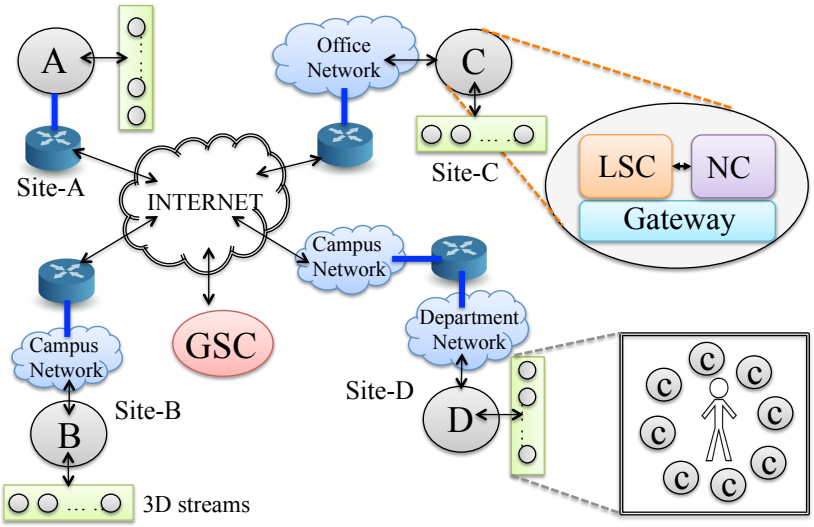


Figure 5.9: (a) 3DTI setup with 4 sites.

Initially, all sites request the same set of remote streams. Due to the notion of stream priority, each view demands only 4 streams from each remote participant (covering 180^0 space). The initial multi-stream overlay is constructed using the topology shown in Figure 5.10. However, during the session run-time, the overlay is updated to meet the view change demands. We compare the performance of OpenSession to a "no OpenSession" case, where the gateway performs the sole data plane functionality without using the network support.

5.7.2 Performance Metrics

To measure the application performance, we run the same 3DTI session with and without the OpenSession control plane. No view change is requested. Each performance value is measured, in percentage, with respect to the measurement collected without using OpenSession (i.e., no OpenSession).

Local bandwidth: OpenSession achieves a significant reduction in bandwidth within the local network. We divide the bandwidth usage into two parts. First, we plot how

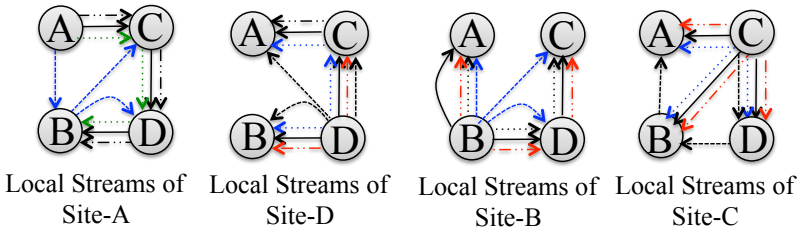


Figure 5.10: Priority-driven multi-stream content distribution overlay graphs for local streams of each site.

much bandwidth we use within the local network between the gateway and the OpenFlow switch at each site. We notice that (in Figure 5.11) Site-B, Site-C and Site-D achieve about 55% reduction in bandwidth for local streams in OpenSession. Second, we plot how much bandwidth we use within the local network for the remote streams. The OpenSession reduces about 35% bandwidth for the remote streams.

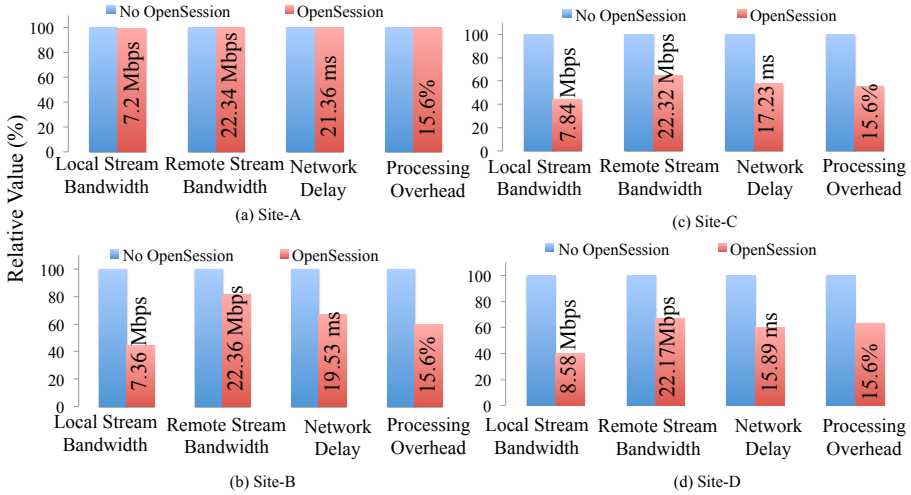


Figure 5.11: (a)-(d) The relative value (with respect to the measurements without OpenSession) of performance metrics in a 3DTI session at different sites.

Processing Load: We measure processing load of the gateway, which includes stream receiving, scheduling and forwarding load of the application. As shown in Figure 5.11, OpenSession lowers stream scheduling and forwarding load at the application layer data plane by offloading part of the forwarding functionalities to the network layer data plane. On average, the OpenSession improves CPU overheads by 42% per site.

End-to-end delay: The gain in network transmission delay is achieved in OpenSession as streams do not traverse the local network for multicast-based forwarding at each multicast hop. We achieve about 45% gain (about 8ms) in the end-to-end network delay (will be higher for wireless end host) for transmitting 3D frames from the source to the destination site.

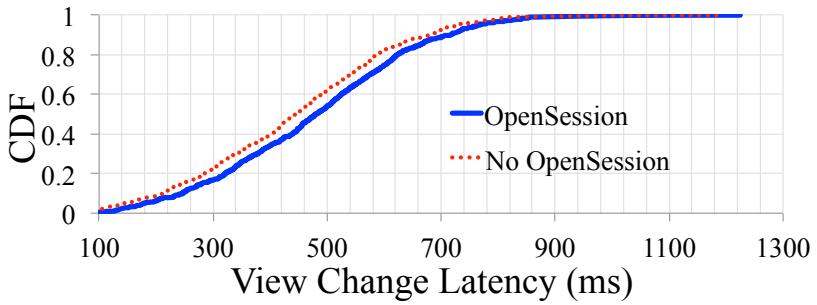


Figure 5.12: View change latency (latency between the sites is 15-20ms).

5.7.3 Impact of View Change

Views are changed according to the Zipf distribution of 20 pre-selected view orientations (22.5^0 apart). 100 view changes are requested from one site (Site-B) within 1 hour at equal interval. Routes for the impacted streams are updated considering constraints RUC (Section 5.5.2). Valid peers (not violating RUC) to update streaming paths are selected randomly. No streams are dropped after session adaptation.

View change latency: View change latency is measured as the difference between the time when a view change is requested by Site-B and the time when Site-B receives new streams according to the view demand. In "no OpenSession" case, after computing the updated overlay at the GSC, the application data plane at the gateway is updated without considering the impact on victim sites. In OpenSession, the view change latency includes the steps shown in Figure 5.6. We plot CDF of view change latency in Figure 5.12 with OpenSession and without OpenSession. The additional overhead in view change by OpenSession control is very small ($<$50ms).

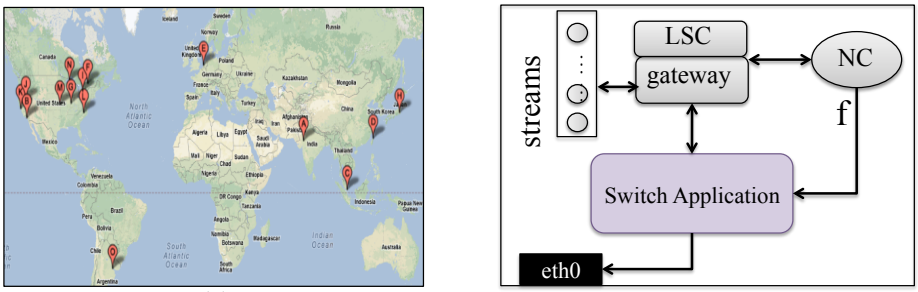
View change resiliency: OpenSession achieves view change resiliency since victim sites are not overloaded or disconnected during view changes. However, in "no OpenSession", if distributed flow tables are updated without maintaining any particular order, packet losses occur at the victim sites. To measure the gain in view change resiliency with OpenSession, we run the same 3DTI session with and without OpenSession. Packets are dropped if disconnectivity occurs in streaming at the victim sites. Table 5.1 shows the total number of packet losses per view change for all streams at all victim sites. OpenSession provides higher resiliency in view change.

Approaches	**Avg**	**Stdev**	**Max**	**Min**
OpenSession	0	0	0	0
No OpenSession	20.35	9.12	45	9

Table 5.1: Number of packet losses at victim sites due to view change

5.8 Evaluation with PlanetLab

We use PlanetLab to evaluate the performance of OpenSession under real Internet artifacts in real-time 3DTI streaming.



(a) (b)

Figure 5.13: (a) Geographical distribution of PlanetLab nodes, and (b) application pipeline at each PlanetLab node.

5.8.1 Evaluation Setup

To run 3DTI in PlanetLab nodes, we collect 15 nodes from different geographical regions (in Figure 5.13(a)). For each 3DTI session, we randomly select 10 nodes. Each node hosts one gateway, LSC and NC as shown in Figure 5.13(b). As we do not have access to the network components of the remote nodes, we develops a switch application (keeps FT in memory) that uses f (from Figure 5.1(b)) to simulate the rule insertion latency. The end-to-end delays are less than 180ms and inbound and outbound bandwidths are within 10Mbps and 250Mbps, respectively. We consider three overlay construction approaches shown below due of their close relation to multi-stream 3D teleimmersive applications.

Random: In the Random approach, a random multi-stream topology is constructed during session initiation. It does not consider stream priority. During view changes, streaming sources are selected randomly.

ViewCast: In ViewCast [4], initial multi-stream overlays are constructed considering the stream priority. During view changes, the higher priority streams (among the impacted streams) are restored (randomly) first. Though ViewCast is a distributed algorithm, we implement it in a centralized. way.

OSM: In OSM [23], the initial multi-stream overlays are constructed using evolutionary optimization to ensure a globally optimized overlay considering stream priority. During view changes, optimal streaming paths are selected for the streams.

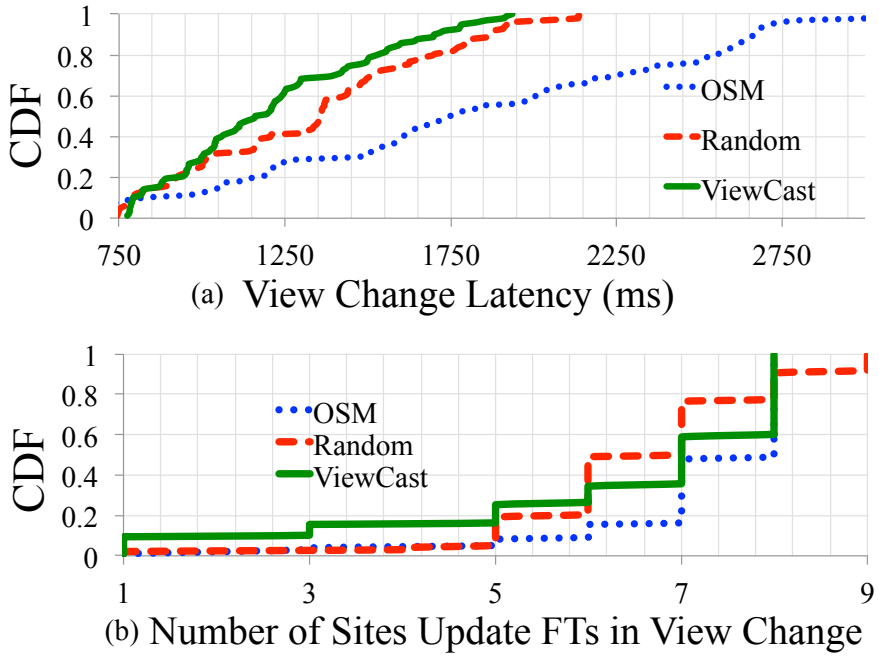


Figure 5.14: For different overlay computation approach in PlanetLab setup, (a) view change latency, and (b) number of sites update flow tables.

5.8.2 Impact of View Change

View change overhead: Figure 5.14(a) shows CDF of view change latency incurred in OpenSession with three different overlay construction algorithms. Though OSM causes larger view change latency, the overhead mainly comes from the optimization process for the computation of the new overlay after a view change is requested. The average overlay computation latencies for OSM, Random and ViewCast are 1.3sec, 870ms and 763ms, respectively. Since the update of flow tables for a view change occurs at all sites in parallel except at the source site, the communication latency is proportional to the two round-trip delays between the participating sites and the GSC. Figure 5.14(b) shows the CDF of number of sites update their flow tables over 100 view changes. Most of the cases, 70% of the sites are required to update their FTs. The number of sites here corresponds to the OpenSession message overhead during a view change. The message overhead of OpenSession protocol is very small since the size of a route update message is 400B, which leads to about 2.8KB overhead per view change.

View change resiliency: To understand the impact of view change resiliency in PlanetLab setup, we run 10 3DTI sessions among the PlanetLab nodes with and without OpenSession. In "no OpenSession", FT rules are updated in two different orderings across the distributed site-gateways when view changes are requested. In the first case (case-I), we update FT rules allowing overloading but no disconnectivity on the streaming path. In the second case (case-II), we update FT rules allowing disconnectivity, but no overloading on the streaming path. Over 1 hour experiment, we modify the number

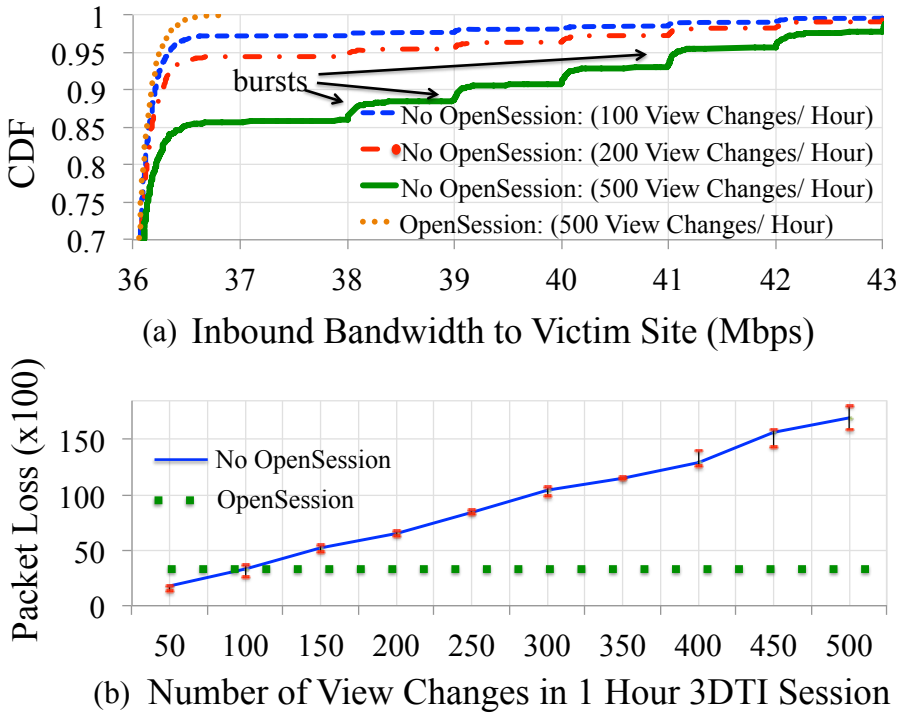


Figure 5.15: Impact of view change frequency on (a) inbound bandwidth to the victim site, and (b) total number of packet loss for all streams during 1 hour.

of view changes to get different view change frequency. We consider Random overlay construction.

Figure 5.15(a) plots the CDF of average inbound bandwidth at the victim sites measured in 5 seconds interval during 1 hour session for case-I. Without OpenSession, overloading creates a sudden burst in the inbound bandwidth of the victim sites. The number of bursts increases with the increase of view change frequency. However, in OpenSession, the inbound bandwidth is fairly constant even with higher view change frequency. Figure 5.15(b) shows the total average number of packet losses at the victim sites for all streams for case-II over 1 hour session. Without OpenSession, the number of packet losses increases linearly with the increase of view change frequency due to the frequent disconnectivity. However, OpenSession does not create any packet loss at the victim sites due to view changes. Similar resiliency can be shown for site failures.

5.8.3 OpenSession Overhead

As we implement NC inside the Floodlight OpenFlow controller, there is an extra processing overhead in the controller. To measure this overhead, we use the *cbench controller benchmarking tool* [94]. We run cbench in *throughput* mode, emulating 1 switch connected to one site. We are able to process 225283 `packet_in` events per second as compared to 241768 in Floodlight without OpenSession. The overhead of NC is only about 6%.

5.9 Discussion and Conclusion

OpenSession protocol adds following features in current real-time collaborative applications (considering Figure 5.11):

Simplified data plane: The separation of data and control plane in OpenSession makes application logic simpler. Furthermore, the partial offloading of application data plane to the network layer reduces multi-stream management and forwarding overhead of the application host (about 42%).

Improved application performance: The splitting of data plane comes with the benefits of lower bandwidth overhead (over 55% for local streams and 35% for remote streams) within the local network (i.e., last mile) and lower network transmission delay (over 45%) in multicast-based streaming.

Seamless session adaptation: OpenSession handles victim sites in session adaptation process and seamlessly updates the running session without introducing any streaming interference (packet loss or traffic burst) at the victim sites. The improvement is shown by a demo video in [95].

Optimized control plane: OpenSession optimizes the flow table size at the OpenFlow switches. The switch controller does not keep any state, and the processing overhead at the controller is very small (about 6%).

We have shown that in addition to the great potential of SDN in data center, or enterprise networking solutions, it can be used very efficiently and successfully for wide area interactive networking applications. OpenSession successfully leverages the network layer OpenFlow functionality to improve application performance in 3DTI.

6 Large-Scale Multi-stream Dissemination

6.1 Introduction

The last two chapters described our efforts in managing session for immersive users. In this chapter, we focus on session management for content streaming towards non-immersive users. Though the computation of multi-stream topology and resource allocation by a central controller (i.e., using GSC as shown in Figure 1.6) works nicely for the immersive users, it does not scale for a large number of non-immersive users. For this purpose, we introduce group session controller or GrSC as shown in Figure 1.6 in the non-immersive domain, which works as a GSC for a group of non-immersive users. GSC is still used for global coordination across the distributed GrSCs.

The large-scale multi-stream and multi-view dissemination of 3DTI contents in the non-immersive session introduce several challenges. A we discussed in Section 5.3.2, bundle of streams generated across the sites at any point in time are highly dependent; so are streams inside a bundle (stream generated from the same site), because they collaboratively represent a consistent virtual scene. Such time dependencies must also be preserved at the non-immersive viewers due to the same reason. Figure 6.1 shows a case, where streams coming from two different immersive users are merged without considering the temporal dependency at a non-immersive user. The output results a view, which represents an inconsistent state of the system. However, keeping the inter-bundle skew (delay difference between dependent bundles) or local inter-stream skew (delay difference between dependent streams inside a bundle) bounded in current Internet is very difficult due to the heterogeneity of dissemination paths taken by the dependent streams in a view. Often, in case of large inter-bundle skew or local inter-stream skew, the lagged streams are dropped to display a consistent scene, even though they consume network and system resources, which causes ineffective resource usage.

Secondly, the large scale 3D multi-stream dissemination introduces a large demand on bandwidth. Each 3DTI stream consumes around 2Mbps to 10Mbps bandwidth, which is exacerbated due to the presence of multiple streams in each view as well as the need to stream multiple views to many viewers, one view from each content producer site. Without proper allocation of bandwidth, it is hard to support a large number of concurrent views as well as viewers in this dynamic environment.

Finally, the 3DTI systems allow viewers to dynamically change their views during run time similar to TV channel switching. However, unlike TV channels, views may contain overlapping streams. Changing views changes the stream subscription of the viewers. Such view dynamics impacts the content dissemination topology and interfere

on-going transmission of already subscribed streams. Due to the nature of live 3DTI streaming, such inference should be low and restored quickly.

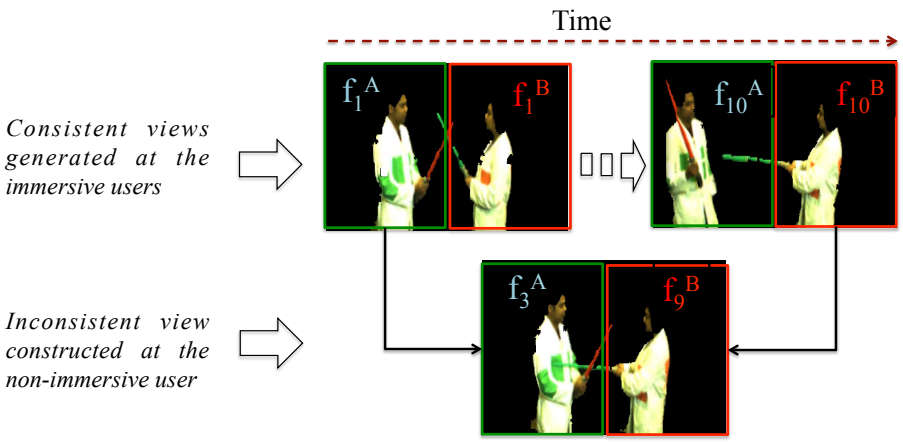


Figure 6.1: Example showing the necessity of preserving multi-stream dependency at the non-immersive viewers. 3D frames f_1^A to f_{10}^A are coming from Site-A and f_1^B to f_{10}^B are coming from Site-B. An inconsistent view are generated at the non-immersive viewers if temporal dependencies are not considered between f_i^A and f_j^B.

Based on the above challenges, we find that there is a need to construct an efficient 3D content dissemination framework that allows non-immersive viewers to subscribe to multiple streams while optimizing the bandwidth resources and preserving the stream dependencies. We present 4D TeleCast [26], a novel 3DTI content dissemination framework that scales to a large number of viewers and provides the functionality of view selection. We call it 4D TeleCast because we consider four dimensions in the tele-immersive space: (1) height, (2) width, and (3) depth of the tele-immersive video streams generated by individual producer sites, and (4) composite view of a viewer to watch these video streams from different locations in a virtual space.

4D TeleCast adapts the hybrid dissemination techniques from [3][96] to balance the impact of centralized load of streaming and overhead of fully distributed management. A CDN (Content Distribution Network) is used to capture and distribute stream contents from the content producers (i.e., immersive users) with P2P routing at the edges (i.e., among non-immersive users). An effective bandwidth allocation algorithm (Section 6.3) is considered at the P2P layer based on the stream priority to allocate inbound and outbound bandwidth at the viewers. A *degree push down* algorithm is used to finally build a P2P overlay that allows maximum number of concurrent non-immersive viewers (Section 6.4). Finally, to preserve the multi-stream dependencies, a *delay-based layering* is introduced below the CDN that bounds the differences in the *end-to-end delay* (the delay between the point when a frame is captured and the point when it is received at the viewer) incurred by the streams inside a requested view. It improves effective bandwidth usage in the system by using delayed receive for the streams with lower end-to-end delay. We use the term ”non-immersive users” and ”viewers” interchangeably in this chapter.

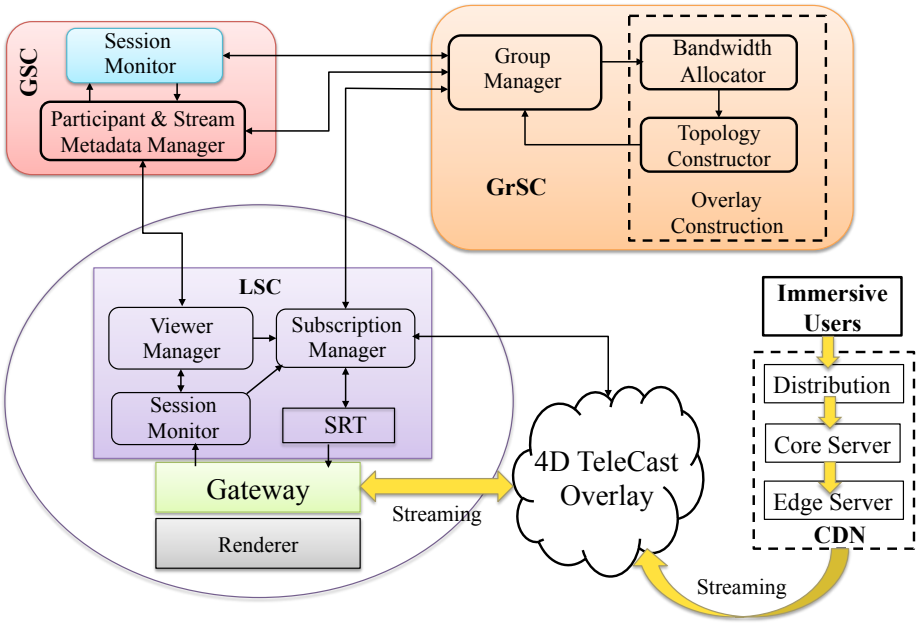


Figure 6.2: 3D TeleCast system components and their interactions.

6.2 System Architecture and Overview

4D TeleCast provides a hybrid overlay dissemination framework consisting of a CDN to transmit 3D contents from immersive users to non-immersive users, and P2P overlays to transmit them within the non-immersive users. The purpose of using hybrid dissemination architecture is that using only CDN servers for streaming is very expensive in terms of deployment and maintenance. On the other hand, a P2P-based architecture requires sufficient number of seed supplying peers to *jump-start* the distribution process and offers less out-bound streaming rate compared to a CDN server [96]. Therefore, the commercial CDN-P2P-based live streaming system (e.g., LiveSky [3]) shows better performance than pure P2P-based live streaming systems [97].

A global session controller (GSC) (as shown in Figure 1.6) node is used to manage the live session of TeleCast content dissemination. To scale GSC, we divide the geographical region into several region-based clusters and assign a group session controller (GrSC) to each cluster that manages the streaming requests of all the viewers inside its own cluster. The session monitor, and participant and stream metadata manager at the GSC work similarly as in the immersive session. For finding the geo-location of the non-immersive viewers, we use a location detector algorithm similar to [98]. Figure 6.2 shows the architecture of TeleCast with different functional components and their interactions. Below we explain them.

6.2.1 Server Side Distribution

Content producers and CDNs are acted as 4D TeleCast media and distribution servers. Several CDN providers are available in market (e.g., Akamai [99] and Amazon Cloud-

Front [100]). Though they do not provide any real-time service guarantee on content delivery, TeleCast aims to use those commercial CDNs for live 4D contents streaming due to the non-immersive nature of the viewers.

The CDN architecture usually contains a storage (called *distribution* in Figure 6.3), where 3DTI contents (3D media frames) are uploaded from the immersive users' gateways. The *core servers* distribute the contents to different *edge servers*. In addition, CDN internally handles load distribution, data replication and data availability across the edge servers so that the viewer requests can be served quickly [101]. Note that, 4D TeleCast uses the CDN as a storage and first layer distribution server and does not require any changes in the architecture or software inside the CDN.

6.2.2 Client Side Distribution

3D TeleCast clients are non-immersive viewers. Each viewer requires a viewer software that includes both the gateway and the renderer as shown in Figure 1.2. To join, the *viewer manager* inside the LSC requests a view with inbound and outbound capacity information. The request first goes to the *participant and stream metadata manager* component at the GSC. GSC then finds the appropriate GrSC from the *session monitor* component based on the viewers geographical region, and forwards the viewer's join request to the *group manager* component of the GrSC to initiate the join process.

Depending on the view information, GrSC group manager computes the list of requested streams and their priorities in the current view, and forwards them to a *bandwidth allocator* along with viewer's inbound and outbound capacity information. The bandwidth allocator allocates viewer inbound and outbound capacity for the requested streams. Due to the bandwidth insufficiency, some of the lower priority stream requests may be dropped. The list of accepted streams in the view is sent to the *topology constructor* component. It creates the final overlay tree (i.e., defines the parents and children) for each accepted stream considering the bound on the end-to-end delay.

Topologies are formed separately for each view group, i.e., the topology formation component groups the viewers depending on the view request. The grouping ensures that the popular view creates enough resources (or seeds) to share and support other viewers with the same view compared to the non-popular views, and does not get interfered by the non-popular views. The algorithm first tries to provision a viewer request from the available viewers watching the same view, if failed, the request is provisioned from the CDN (provided the CDN has unused outbound capacity). The overlay information is then sent to the viewer and to the parents of the viewer to modify their overlay routing tables. The goal of bandwidth allocation and topology formation (compositely we call them overlay construction) is to build an optimal overlay that maximizes the number of accepted streams in the overlay and minimizes the CDN usage. The detail algorithms for overlay construction are given in Section 6.3.

Once, the LSC at the joining viewer receives the topology information (list of parents and children) with the list of accepted streams from the GrSC, it updates its session routing table (SRT). The fields of SRT are already given in Section 4.4.2. However, 4D

TeleCast considers an additional field in the SRT, which we will describe in the next paragraph. Though the overlay structure ensures the delivery of all accepted streams in the view request, it does not bound the inter-stream delay inside the view, which creates the *view synchronization* problem. The phenomena is explained in Figure 6.3(a), where the viewer u is provisioned to receive streams S_6^A, S_7^B, S_6^B by its LSC. Here, d_{buff} defines the time a frame of a stream stays in the buffer after it is received. As we see in the figure, when a frame of S_6^B is received, the correlated frame (captured at the same time) of S_6^A is already been discarded from the buffer. Since the end-to-end delay between S_6^B and S_6^A is higher than d_{buff}, viewers can not display them together at the renderer. This creates a lower quality view at u even though the network resources are consumed for the higher quality view. Here, we assume that the maximum unnoticeable inter-stream skew, $d_{skew} = 0$.

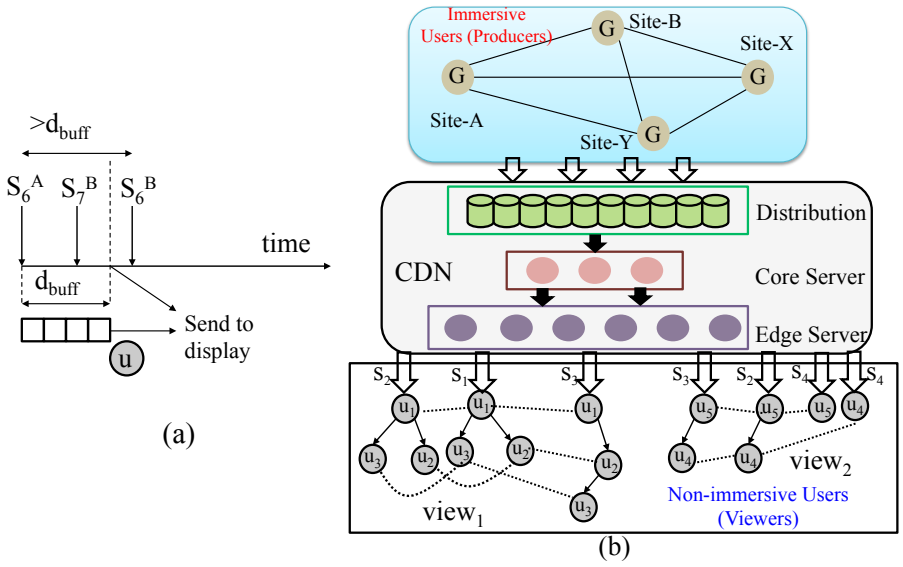


Figure 6.3: Examples of (a) 3DTI view synchronization problem, and (b) 4D TeleCast dissemination and streaming architecture, where u_1, u_2 and u_3 request $view_1$=[S_1, S_2 , S_3] and u_4 and u_5 request $view_2$=[S_2, S_3, S_4].

To solve this view synchronization problem, the *subscription manager* in the viewer LSC computes the *subscription point* for each accepted stream. The subscription point of a stream defines the position at the cache of the parent viewers from where the parents should forward the stream. Therefore, SRT in 4D TeleCast includes subscription point in viewer cache in addition to the 4 fields mentioned in Section 4.4.2. It eventually introduces *delayed receive* for some of the streams so that the dependent streams are always present in the buffer. To understand this stream subscription point in the overlay in terms of delay, we introduce a delay layer hierarchy. The details of the delay layer hierarchy architecture and a stream subscription algorithm to solve the view synchronization problem are shown in Section 6.4. However, if the delayed receive violates the maximum allowed end-to-end delay for a stream, the request for that stream is dropped and the resources are made available to use by the other viewers.

After the view synchronization is computed, the stream subscription component at the viewer site sends the subscription point of the accepted streams to the corresponding parents and children. When the child-viewers receive the subscription information, they update their synchronization point accordingly in their SRTs. When the parent viewer receives the stream subscription requests, it updates the subscription point in its SRT. After the SRTs are updated, the gateways start streaming from the subscription point of its cache for each requested stream.

However, due to the frame loss, congestion or other network events, the inter-stream delay may change. Therefore, the session monitor at the GSC periodically analyzes the monitoring data (via the viewer management component) and updates the delay layer hierarchies and the stream subscription layers. It also handles overlay fault tolerance due to the viewers arrivals and departures. An example of the 4D TeleCast distribution structure is shown Figure 6.3(b).

View Change: When a viewer requests a view change, to respond quickly, 4D TeleCast supports the streams in the new view instantaneously from the CDN. Therefore, the bandwidth allocation and topology formation component (in Figure 6.2) at the LSC, when detect a view change, initiate two parallel join processes. The first process serves all streams in the request directly from the CDN and the second process initiates a normal join explained above. Once the second process is done, the viewer is switched to the overlay constructed by the second process. Since the second process runs in the background, the overall approach reduces the response time for a view change. Similar algorithm is used to re-attach the *victim viewers* (who get disconnected from the streaming tree due to the view changes) into the streaming overlay. We explain the view change process in Section 6.4.

6.3 Multi-stream Overlay Construction

4D TeleCast provides a hybrid overlay dissemination framework consisting of a CDN to transmit 4D contents from producers to content viewers, and P2P overlays to transmit them within the viewers. The purpose of using hybrid dissemination architecture is that using only CDN servers for streaming is very expensive in terms of deployment and maintenance. On the other hand, a P2P-based architecture requires sufficient number of seed supplying peers to *jump-start* the distribution process and offers less out-bound streaming rate compared to a CDN server [96]. Therefore, the commercial CDN-P2P-based live streaming systems (e.g., LiveSky [3]) show better performance than pure P2P-based live streaming systems [97]. Using the hybrid structure, the overlay construction problem can be defined as follows.

6.3.1 Overlay Construction Problem

The overlay construction problem considers three constraints (bandwidth, delay and number of accepted stream constraints), and one optimization goal.

Bandwidth Constraint: Each viewer u has limited total inbound bandwidth (C^u_{ibw}) and limited total outbound bandwidth (C^u_{obw}). The CDN also defines a bounded inbound and outbound capacity (C^{cdn}_{ibw} and C^{cdn}_{obw}, respectively) that can be used in the 3DTI session. Due to the limited number of producers sending content to the CDN, we assume C^{cdn}_{ibw} bound is always met.

Delay Constraint: Even though, the viewers are not immersed into the 3DTI space, they impose a delay bound for live streaming. d_{max} defines the maximum delay from the time when a 3D stream is captured at the producer site until the time when it is displayed at the viewer's display.

Number of Accepted Stream Constraint: As we mentioned before, to accept a viewer request, at least one stream (i.e., the most priority stream) from each producer site included in the view should be served to the viewer, i.e, for n number of producer sites, $N^u_{accepted} \geq n$.

Optimization Goal: Due to the delay and bandwidth constraints listed above, we cannot guarantee that all stream requests can be satisfied. The metric we wish to maximize first is the *acceptance ratio (AR)* for all requests in the system. Suppose, the number of streams that viewers request is N_{total}; among which $N_{accepted}$ are accepted due to the resource constraints, then we have $AR = \frac{N_{accepted}}{N_{total}}$. However, we also want to minimize the *CDN outbound capacity usage (obw_{cdn})* in the system so that cost of distribution becomes less (the use of 1GB traffic in Amazon CloudFront [100] CDN costs $\$0.18$). Formally, the overlay construction problem aims to build a dissemination architecture from producers to the viewers that maximizes the total acceptance ratio with minimum CDN outbound capacity usage such that $N^u_{accepted} \geq n$, $ibw_u \leq C^u_{ibw}$, $obw_u \leq C^u_{obw}$, $d^u_{S_i} \leq d_{max}$ and $ibw_{cdn} \leq C^{cdn}_{ibw}$ $\forall u \in U$, where ibw_u and obw_u are the inbound and outbound bandwidth usage of u, $d^u_{S_i}$ is the end-to-end delay of stream S_i from the producer to u and U is the set of all connected viewers in the overlay.

We provide a heuristic solution since an optimization problem in multicast with two or more constraints is NP-C [49]. We divide the solutions into two parts: *bandwidth allocation* and *overlay formation*.

6.3.2 4D TeleCast Solution

The proposed solution for multi-stream overlay construction is divided into three parts: 1) inbound bandwidth allocation, 2) outbound bandwidth allocation, and 3) topology formation. Below we discuss them.

Inbound Bandwidth Allocation: The process starts by performing bandwidth allocation on the viewer's inbound capacity. Streams are assigned required inbound bandwidth at the viewer in the order of their priorities provided that two conditions are met; 1) there is enough inbound bandwidth left for allocation at the viewer, and 2) the P2P layer or CDN has enough outbound bandwidth to support the stream. If any of these two conditions are violated, the lower priority streams are not assigned any bandwidth and they are removed from the view request. Suppose, a viewer u requests

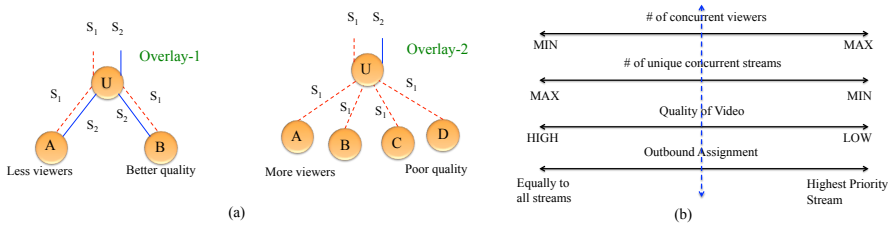


Figure 6.4: (a) Example of different outbound allocations, (b) Tradeoffs among different parameters impacted by the outbound bandwidth allocation.

for view $view_m = \{S_1, S_2, \cdots S_n\}$ containing n streams from the participating immersive sites with priority $S_1 > S_2 > \cdots > Sn$, $abw_{Si}^{view_m}$ is the current available outbound bandwidth for stream S_i for view $view_m$ and bw_{S_i} is its required network bandwidth. To allocate viewer's inbound bandwidth, LSC assigns bandwidth bw_{S_i} to stream S_i if $bw_{S_i} \leq abw_{S_i}^{view_m}$ for first j streams in the order of their priorities such that $\sum_{i=1}^{j} bw_{S_i} \leq C_{ibw}^{u} < \sum_{i=1}^{j+1} bw_{S_i}$. Hence, the list of accepted streams in $view_m$ becomes $S_1, S_2, \cdots S_j$. If $j = n$, then all streams are accepted.

If the number of accepted streams is less than the total number of participating sites (i.e., at least one stream from each site), then the viewer's request is rejected. Otherwise, the GrSC performs bandwidth allocation on the viewer's outbound capacity.

Outbound Bandwidth Allocation: Once the inbound bandwidth is allocated, the GrSC allocates outbound bandwidth only for the set of streams accepted in the previous step. The outbound allocation is critical because if we assign outbound bandwidth to only the highest priority stream of each site, we can support maximum number of viewers but with lower media quality. However, if we assign bandwidth equally to all streams, we get limited number of viewers but with better media quality. The phenomena is shown in Figure 6.4 (a). Overlay-1 supports less number of viewers but with better view quality (higher number of accepted streams), while Overlay-2 supports higher number of viewers with lower media quality (lower number of accepted streams). Therefore, we need a tradeoff in the outbound bandwidth assignment, where we can support sufficient number of viewers with good quality. Figure 6.4(b) shows the corresponding tradeoff among media quality, number of concurrent accepted viewers, outbound assignment and the number of concurrent accepted streams. Our goal is to maintain the overlay in the middle of the tradeoff lines (shown dotted). Though, the problem is complicated it turns out that using a round-robin allocation of outbound bandwidth in the order of stream priority can solve it. Below we discuss the algorithm for outbound bandwidth allocation.

The outbound allocation starts with allocating the bandwidth for S_1 and continues until S_j similar to inbound allocation, but it rounds up, i.e., it starts allocating from S_1 again if there is enough bandwidth left after assigning the outbound bandwidth to other lower priority streams. It ensures that even with the bandwidth limitation, the highest priority streams in a view has higher probability to be served compared to the lower priority streams. Basically, if $S_i > S_j$ for a view $view_m$, then at any point in time

$obw_{S_i}^{view_m} \geq obw_{S_j}^{view_m}$, where $obw_{S_i}^{u}$ is the allocated outbound bandwidth for stream S_i at viewer u. The out-degree link for S_i at u is computed by $oDeg_{S_i}^{u} = \lfloor \frac{obw_{S_i}^{u}}{bw_{S_i}} \rfloor$. After the bandwidth allocation, the allocated out-degree is used to connect the viewers to the streaming tree.

Topology Formation: For building the overlay topology for each accepted stream in a view, GrSC only considers the viewers with the similar view request as the peers of the P2P layer. A degree push down algorithm is used over the allocated inbound and outbound bandwidth. The goal is to improve the depth of the tree by constructing a flatter tree. Also, it maximizes the number of viewer nodes that can be accepted in the tree within the fixed height by pushing higher out-degree viewers towards the root (flatter trees).

Input: u, $oDeg_{S_i}^{u}$
Algorithm:
set Equeue $Q1 = \{root\}$;
repeat
  $Q2 = Q1$;
  for all $z \in Q2$ **do**
    if ($oDeg_{S_i}^{u} > oDeg_{S_i}^{z}$) or ($oDeg_{S_i}^{u} == oDeg_{S_i}^{z}$ and $C_{obw}^{u} > C_{obw}^{z}$) **then**
      replace z by u; /* bring the higher degree up in the tree */
      $\boldsymbol{Child_{S_i}^{u}} = \boldsymbol{Child_{S_i}^{u}} \cup z$.;
      return 1;
    end if
    $Q1.dequeue(z)$; /* for level-order traversal */
    for all each child $ch \in \boldsymbol{Child_{S_i}^{z}}$ **do**
      $Q1.enqueue(ch)$
    end for
  end for
until ($Q1$ is empty)
return 0;

Algorithm 3: Degree Push Down Algorithm

Algorithm 3 explains the 4D TeleCast degree push down algorithm. The algorithm is executed for each accepted stream in the view request. It uses the viewer id u and its out-degree $oDeg_{S_i}^{u}$ (computed in the previous section) as inputs. We use two priority queues $Q1$ and $Q2$ that store the viewers in ascending order of their out-degrees at each level of the streaming tree. $\boldsymbol{Child_{S_i}^{u}}$ defines the set of child viewers for stream S_i at u. For empty child we put out-degree -1. Starting from the root, the algorithm looks for viewers z such that ($oDeg_{S_i}^{u} > oDeg_{S_i}^{z}$) or ($oDeg_{S_i}^{u} == oDeg_{S_i}^{z}$ and $C_{obw}^{u} > C_{obw}^{z}$). If found, it is replaced by u (which also updates $\boldsymbol{Child_{S_i}^{u}}$) and the replaced viewer is added as another child of u. If the algorithm returns 0 for a stream, the stream is requested from the CDN provided that current usage of the CDN is less than the maximum capacity allowed, otherwise the stream is rejected. The overlay ensures that viewers with higher outbound bandwidth receive streams with lower end-to-end delay, which provides an incentive to the viewers to engage more bandwidth in their session.

The property can be described as follows.

> **Overlay property**: *If viewers u_1 and u_2 request for the same view $v = [S_1, S_2, \cdots S_n]$ under the same LSC, then if u_1 is in higher layer than u_2 in the same streaming tree for one stream, then it is in higher layer compared to u_2 for all other streams.*

We can easily prove it. Since, the outbound bandwidth is allocated in a round-robin manner in order of stream priorities and the priorities are always fixed inside a group (defined by view), the viewer with higher bandwidth always assigns higher outbound bandwidth to a stream compared to the other viewers with lower bandwidth inside the same view group. Moreover, the degree push down algorithm always puts the viewer with higher bandwidth closer the root. Therefore, for a certain view group, the viewers with higher out-degree becomes closer to the root compared to the viewers with lower out-degree and this is true for all the streaming trees they are subscribed to.

Once the topology construction is completed for all accepted streams, GrSC sends the set of accepted streams and the overlay information back to the viewer. The overlay information includes the parent ($Parent^u_{S_i}$) and children ($\boldsymbol{Child^u_{S_i}}$) IDs along with their end-to-end delay and layer information for each accepted stream.

6.4 View Synchronization

6.4.1 View Synchronization Problem

The goal of view synchronization is to preserve the multi-stream dependencies at the viewer with the given bandwidth constraints. If the dependencies can not be preserved for a stream, then the bandwidth used by that stream should be made available to be used by other viewers. Suppose, at viewer u, the set of accepted streams in the request $view_m$ is $\{S_1, S_2, \cdots S_j\}$. $d^u_{S_i}$ is the end-to-end delay of stream S_i at u perceived by the overlay structure. Though, the overlay construction problem bounds $d^u_{S_i}$ (where $d^u_{S_i} << d_{max}$), it does not provide any bound on $|d^u_{S_i} - d^u_{S_k}|$ $(1 \leq i, k \leq j)$. If the viewer maintains a buffer of length d_{buff} for each requested stream, to represent a synchronous view, the following constraint must be fulfilled: $|d^u_{S_i} - d^u_{S_k}| \leq d_{buff} + d_{skew}$, where d_{skew} defines visually allowed maximum inter-stream skew (i.e., the maximum unnoticeable inter-stream skew at the display). For ease of explanation, we use $d_{skew} = 0$.

6.4.2 4D TeleCast Solution

To solve view synchronization problem, viewers first need to understand the stream end-to-end delay for the list of accepted streams after joining into the streaming overlay. Therefore, we introduce delay layer hierarchy in the P2P dissemination layer. We modify the viewer buffer architecture to support this layer hierarchy, and finally we show how we use stream subscription to solve the view synchronization problem.

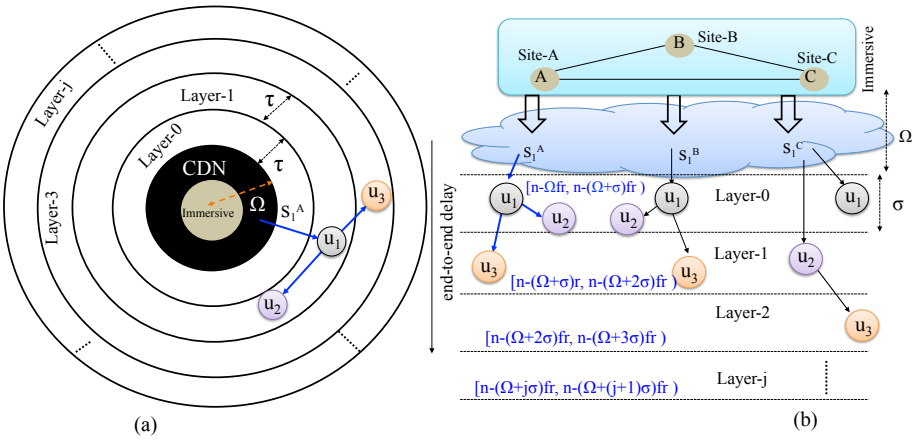


Figure 6.5: (a) Tele-Cast layering architecture, (b) Example of delay layering showing the distribution of frame numbers at different layers at time t.

Delay Layer Hierarchy: The purpose of delay layering is to understand the streams' position at the viewer in terms of end-to-end delay in the overlay. Each layer is identified by the delay value and bounded by a delay duration called σ. We define $\sigma = \frac{d_{buff}}{\kappa}$, where $\kappa \geq 2$. κ defines the layer width. Viewers at the higher layers (with lower layering index) for a stream receive frames with lower delay and the viewers at the lower layers (with higher layering index) receive frames with higher delay. A viewer can be in multiple layers; one layer for each requested stream. Therefore, the layering architecture can be represented as several concentric circles (one for each layer) with the producer and the CDN in the center as shown in Figure 6.5(a).

Definition of Delay Layer: To define formally, suppose, Ω indicates the maximum delay a frame takes to get delivered at any viewer via CDN after it is captured at the producer. Therefore, according to the delay layer architecture, the viewers at Layer-y receive streams with end-to-end delay bounded by $[\Omega+y\sigma, \Omega+(y+1)\sigma)$, where $y \geq 0$. An example of delay layer hierarchy for viewers u_1, u_2, and u_3 requesting $view_1$ is shown in Figure 6.5(b). Depending on the end-to-end delay, u_3 is in Layer-1 for stream S_1 and S_2, but in Layer-3 for stream S_3.

If the frame rate is fr for stream S_1^A, according to the layer architecture, at time t, viewers at Layer-y receive frames with frame numbers between $[n1, n2)$, where n is the latest captured frame number at the producer for stream S_1^A, $n1 = n-(\Omega+y\sigma)fr$, $n2 = n-(\Omega+(y+1)\sigma)fr$. Figure 6.5(b) shows the corresponding layering hierarchy for $y \in \{0, 1, 2\}$.

How to Compute Delay Layer for a Stream: When a viewer is added into the dissemination overlay, the layer (i.e., the lowest layer index) of the viewer for a requested stream (interchangeably we term it as the layer of the requested stream) is computed using the end-to-end delay of that stream at its *parent* in the overlay (from which the viewer receives the requested stream), the processing delay inside the parent (due to internal processing and buffering) and the network propagation delay from the parent

to the viewer. If d_{prop} defines the network propagation delay between u and its parent ($Parent^u_{S_i}$) for stream S_i, $d_{S_i}^{Parent^u_{S_i}}$ is the end-to-end delay of stream S_i at $Parent^u_{S_i}$, and ω indicates the processing delay at the parent, then the layer of viewer u for stream S_i ($Layer^u_{S_i}$) is computed as follows:

$$Layer^u_{S_i} = \lfloor \frac{d_{S_i}^{Parent^u_{S_i}} - \Omega + d_{prop} + \omega}{\sigma} \rfloor \quad (1)$$

In our evaluation, we assume $d^{CDN}_{S_i} + d_{prop} + \omega = \Omega$ for the viewers with CDN parents, i.e., the summation of end-to-end delay from the producers to the CDN and the CDN to its first children take approximately the constant time for each stream and it is equal to Ω. Therefore, the viewers who receive streams directly from the CDN can always achieve the highest layer, Layer-0. However, this constraint can be easily relaxed by considering separate Ω value for each producers' stream.

How to Modify Delay Layer for a Stream: Layer modification at the viewer for a particular stream can be done simply by requesting frames back in time or ahead in time from the parents. However, due to the bound on the propagation and processing delay from the parents, the viewers cannot decrease the layer indexes (i.e., move to the higher layer) from the value measured by Equation 1. If a viewer is at Layer-y for stream S_i, at any time t, it can switch to Layer-x by requesting the streams starting from the frame number n' from its parent, where n' is defined as follows:

$$n' = n - (\Omega + (x+1)\sigma)r + (d_{prop} + \omega)r + d_{prop}r + \Re \quad (2)$$

Here n is the latest frame number at the producer at time t (collected from the GSC monitoring), r defines the frame rate (also collected from the GSC monitoring component) and $\Re$ generates an offset between $[0, \sigma r]$. Using $\Re$, we can chose viewers' position inside the desired layer boundary. The term $(d_{prop} + \omega)r$ in Equation 2 considers the total propagation delay from the parent and the third term $d_{prop}r$ is added since the request to update the stream layer takes additional d_{prop} time to get delivered to the parent. The parent then streams at the media rate starting from n'.

Since, the change in delay layer requests frames of different time period from the parents of the overlay tree, viewers needs to modify their buffering structure to cache 3D frames. Below we discuss the 4D TeleCast viewer buffer architecture.

Extension of Viewer Buffer: We extend the single-stream based buffering architecture [102, 103, 104] used in PPLive and CoolStreaming for multi-stream scenario. Each viewer maintains a local buffer at the gateway. It meets three purposes: 1) hides intra-stream jitter and delay in media playback, 2) hides inter-stream skew for media synchronization, and 3) allows peer sharing of media streams. For simplicity, we consider separate local buffers for different streams. The architecture of a viewer's local buffer is shown in Figure 6.6, where the viewer is subscribed to a view with two media streams: S_1, and S_2. At the *Media Playback Point (MPP)* in the figure, the renderer picks up the synchronized frames (where the difference between the origin timestamps is less than

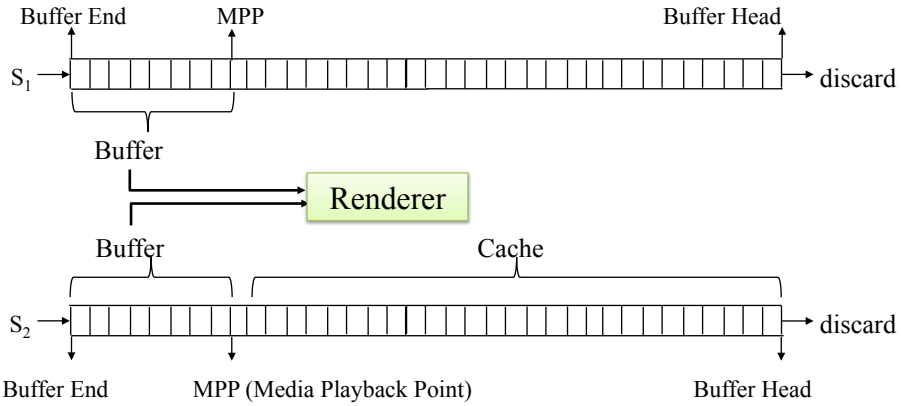


Figure 6.6: Viewer's local buffer architecture in 3DTI TeleCast.

d_{skew}) of S_1 and S_2 from the respective buffers (between MPP and buffer end) and sends them to the display. There can be a separate playback buffer inside the display for smooth playback, which we ignore in our analysis. This is a rather conservative assumption that underestimates the system performance. For the rest of the paper, we use the term "buffer" to indicate the part of the local buffer from buffer end to the MPP and "cache" to indicate the part from MPP to the buffer head. d_{cache} is the caching delay and $d_{buff} << d_{cache}$.

The frames stored in the buffer and cache are both available to support other viewers, while the frames stored in the buffer are only used in local media playback at the renderer. At this point, we propose our first property about the delay layer hierarchy.

> **Layer Property 1:** *A viewer u with end-to-end delay $d^u_{S_i}$ can support S_i from Layer-$\lfloor \frac{d^u_{S_i}-\Omega+d_{prop}+\omega}{\sigma} \rfloor$ to Layer-$\lfloor \frac{d^u_{S_i}-\Omega+d_{prop}+d_{cache}+d_{buff}+\omega}{\sigma} \rfloor$ to any child viewer at distance d_{prop}.*

In case of CDN parents, the distribution storage is very large and hence we assume that the CDN can share any layers of streams to its direct children. This layer property can be easily proved by the definition of delay layer hierarchy.

For simplicity, in our current work, we assume $d_{cache} = d_{max} - \Omega - d_{buff}$. Since, d_{max} is the maximum possible end-to-end delay at the viewers, the value of maximum acceptable layer index is bounded by $\frac{d_{max}-\Omega}{\sigma}$. Therefore, this value of d_{cache} ensures that any viewer can support any acceptable layers of its child viewers into the TeleCast overlay streaming tree.

Stream Subscription: The stream subscription process works locally at each viewer after it joins into the overlay. It includes two steps: 1) finding the layer index for each accepted stream, and 2) bounding the differences in layer indexes so that the delay differences are bounded by d_{buff}. At this point, we present our second property about the delay layer hierarchy.

> **Layer Property 2:** *A viewer u can render dependent streams $(S_1, S_2, \cdots S_n)$ synchronously at the display if the differences in the layering indexes for*

those streams are less than or equal to κ, i.e., $|Layer^u_{S_i} - Layer^u_{S_k}| \leq \kappa$, where $i, k = 1, 2, \cdots n$.

We can easily prove this property. According to the layer definition, if $|Layer^u_{S_i} - Layer^u_{S_k}| \leq \kappa$, the difference between the end-to-end delay of stream S_i ($d^u_{S_i}$) and the end-to-end delay of stream S_k ($d^u_{S_k}$) at u must be bounded by $\kappa\sigma$ (where σ is the size of each layer). Also, we define $\sigma = \frac{d_{buff}}{\kappa}$. Therefore, by combining these two equations, $|d^u_{S/-i} - d^u_{S_k}| \leq \kappa\sigma \leq d_{buff}$. As we mentioned before, if the inter-stream delay can be bounded by d_{buff}, then the renderer can pick the dependent frames from the respective buffers and display a consistent virtual space.

After a viewer joins the overlay structures for j streams, it computes the minimum layer indexes $Layer^u_{S_i}$ for each stream S_i using Equation 1 ($1 \leq i \leq j$). If $|Layer^u_{S_i} - Layer^u_{S_k}| > \kappa$, where $1 \leq i, k \leq j$ and $i \neq k$, then $|d^u_{S_i} - d^u_{S_k}| > d_{buff}$, which violates the view synchronization constraint. To bound the differences by κ, the viewer first finds the maximum layer index $Layer^u_{min}$=max($Layer^u_{S_1}$, $Layer^u_{S_2}$, $\cdots$, $Layer^u_{S_j}$). The updated layer index of each stream S_i is then computed as: $Layer^u_{S_i}$ = max($Layer^u_{S_i}$, $Layer^u_{min}$-κ). We call this process a *layer push-down*. In case of the layer push-down of a stream, the request for streaming are sent to the parent by computing the subscription frame number using Equation 2, otherwise the parents are requested to send frames from their buffer end.

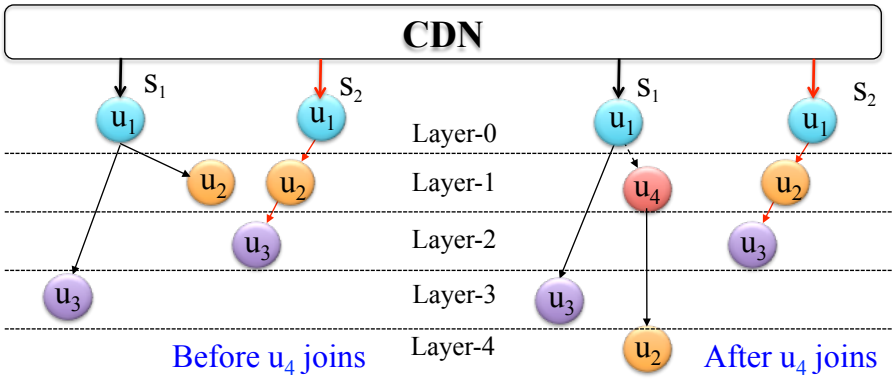


Figure 6.7: Example of push-down when a new viewer joins.

Next, the viewer sends the end-to-end delays to each child in $\boldsymbol{Child^u_{S_i}}$ for each forwarded stream S_i. Once the child viewer ch receives this information, it computes the delay layer index (x) that it can achieve using Equation 1. If $x > Layer^{ch}_{min}$ (the maximum layer index in ch), a new subscription process is started since the delay bound may be violated. However, if $x \leq Layer^{ch}_{min}$, then no layer subscription process is required, because the parent viewer is still able to support ch with its current layer index of the stream.

It is important to note that the layer modification (if any) in the push-down process at the child viewer also needs to be propagated to its children, who may initiate another subscription process. Therefore, when a new viewer joins, the overlay may initiate a

chain of subscription processes. Figure 6.7 shows an example of layer modification in viewer u_2 when a new viewer u_4 joins. Due to the layer modification, the layer bound (κ=2) is violated. So, a layer push-down is required for stream S_2 in u_2. However, we can easily prove that due to the nature of overlay construction, it will not create any *cyclic impact*, which means that if a viewer u_1 starts the initial subscription process in the chain, it does not receive the layer update request along this chain.

Also during the layer push-down using Equation 2, we apply $\Re = \sigma \times r$ (i.e., position the children at the top of the modified layer boundary) so that the push-down fades out (reduce the number of layer modification) in the subsequent children. If the push-down causes a viewer to receive end-to-end delay of a stream higher than d_{max}, either the stream is dropped or the viewer is re-joined in the overlay depending on the importance of the stream.

6.4.3 Impact of Network Dynamics

The value of κ defines the layer width as well as the number of layers to consider within the bounded end-to-end delay d_{max}. If κ increases, the number of layer increases and hence, the layer width decreases. On the other hand, the smaller the value of κ, the lower the number of layers and wider the layer width. Figure 6.8 shows the impact of κ on the layer width and number of layers.

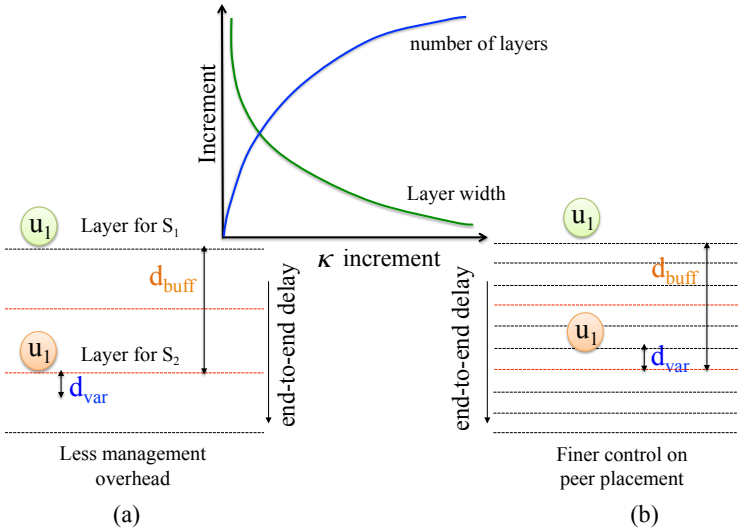


Figure 6.8: Impact of κ on layer width and layer count in the delay layer hierarchy. Examples showing the placement of a non-immersive user u_1 for stream S_1 and S_2 for (a) small κ, and (b) large κ.

If the layer width is narrow, the control over the viewer placement on the hierarchical layering architecture becomes more precise. The precise placement of viewers is important when the end-to-end delay varies over time. For example, in Figure 6.8 (a), the variance of d_{var} causes a violation in delay difference ($> d_{buff}$) between the dependent streams (S_1 and S_2) since the placement of u_1 for S_2 is at the edge of its own

layer. However, in Figure 6.8(b), by placing u_1 precisely in the delay layer, we can keep delay difference bounded even with the variance of d_{var}. Therefore, by increasing the value of κ, we can overcome the delay variance that may cause dependency violation among the dependent streams. However, more layers may cause higher management overhead, which motivates us to keep the κ value small. In our implementation, we show the most of the times κ=2 works good. However, with the delay variation we need to increase κ to ensure high utilization of the effective network bandwidth.

6.5 Performance Evaluation

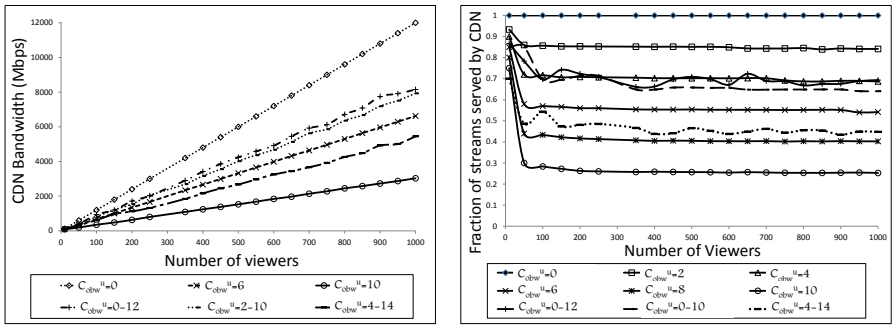
6.5.1 Experimental Setup

We evaluate 4D Tele-Cast using a discrete event simulator. For the experimental setup, we use configurations of system and application components from TEEVE [4], an advanced 3DTI system. We use 2 producers with 8 camera streams at each site representing the content producers. We simulate a CDN that creates minimum end-to-end delay of 60sec (i.e., $\Omega = 60$sec) from immersive sites to the non-immersive viewers. The number of viewers are varied from 10 to 1000. The delay between them are obtained from $4-$hours PlanetLab traces [39]. For each producer stream, we use traces collected from a TEEVE session [4], where two remote participants virtually fight with each other using light sabers. Each stream is bounded by 2Mbps bandwidth requirement. Each viewer requests a view that includes 6 streams; 3 from each producer. We assume each viewer has 12Mbps inbound bandwidth, but the outbound bandwidth (C^u_{obw}) varies from 0 to 14Mbps. The acceptable end-to-end delay at the viewer is bounded by 65 sec (i.e., $d_{max} = 65$sec). The buffer size is 300ms and the cache size to allow peer sharing is 25sec. We fix $\kappa = 2$. We use 1 GSC with 5 GrSCs. We fix layer size $\sigma = 150$ms. The depth of P2P tree is bounded by 10.

6.5.2 Performance of Overlay Construction

The goal of solving overlay construction problem is to maximize the acceptance ratio (ρ) and minimize the CDN usage. We change the viewers outbound bandwidth from 0 to 14Mbps. For different values of outbound bandwidth, Figure 6.9(a) shows the CDN bandwidth requirements to support all requested streams (i.e., to achieve $\rho = 1$). When the viewers do not provide any outbound bandwidth, all requests are served from the CDN (case when $C^u_{obw} = 0$). Most of the cases, the joining viewers provide some bandwidth to allow forwarding. As the figure shows, even if the viewers contribute about 0 to 12Mbps bandwidth uniformly, the required bandwidth from the CDN to support all requests using the hybrid structure is around 6000Mbps. We allocate this amount of outbound bandwidth to the CDN for rest of the experiments.

To understand how much savings in cost we make in 4D TeleCast due to the priority based bandwidth allocation and degree push down based overlay construction, we plot the fraction of requests served by CDN while varying the viewers outbound bandwidth



(a) CDN bandwidth requirements

(b) Fraction of requests served by CDN

Figure 6.9: Performance of 4D TeleCast overlay construction and content distribution. CDN capacity is bounded to 6000Mbps in (b).

in Figure 6.9(b). When the viewers' bandwidth varies between 4 to 14Mbps or each viewer allocate at least 8Mbps outbound bandwidth, about 55% or higher requests are served from the P2P. Hence, the 4D TeleCast saves distribution cost by saving the CDN resources.

If we bound the CDN capacity, some of the viewer requests may not go through due to the lack of bandwidth availability. To understand the performance of the 4D TeleCast in terms of the number of accepted requests with $C_{obw}^{cdn} = 6000$Mbps, we plot the acceptance ratio in Figure 6.10. When the viewers do not provide any outbound bandwidth the acceptance ratio becomes low. When the viewers contribute outbound bandwidth, in most of the cases, the acceptance ratio is very high. The system achieves perfect acceptance ratio when the viewers' bandwidth varies between 4 to 14Mbps or each viewer allocates at least 8Mbps outbound bandwidth.

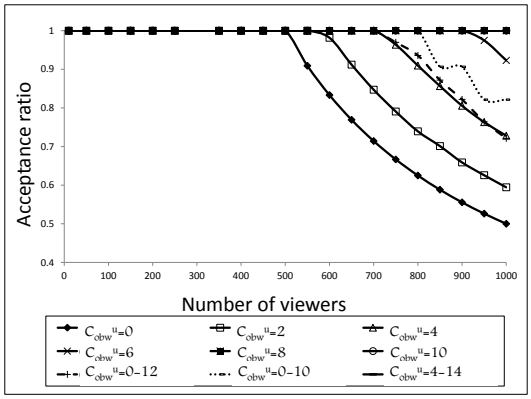
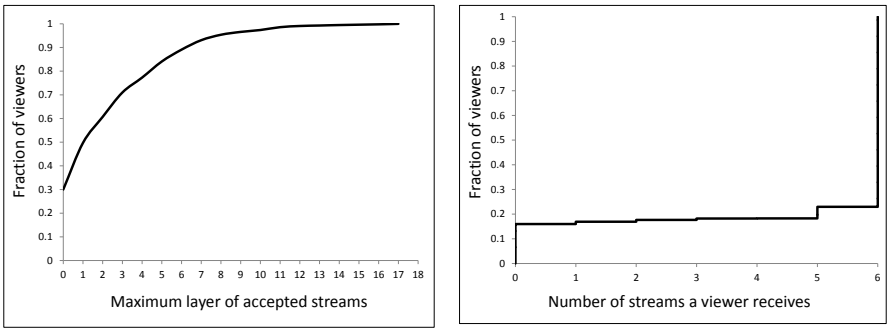


Figure 6.10: Acceptance ratio of 4D TeleCast for different bandwidth distribution. CDN capacity is bounded to 6000Mbps.

6.5.3 Performance of Stream Subscription

In this section, we show the performance of view synchronization using the delay layer hierarchy. We uniformly assign each viewer an outbound bandwidth between 0 to 12Mbps. The minimum subscription layer of the accepted streams at each viewer is shown in Figure 6.11(a). About 30% of the viewers are in Layer-0 and 80% of the viewers are watching the contents in Layer-4 or less. However, not all viewers receive 6 streams requested in the view. In case of bandwidth limitation, the lower priority streams are dropped. However, the inter-stream delay among the accepted streams are always less than 300ms due to the bound on the layering.



(a) Layers of viewers (b) Number of accepted streams

Figure 6.11: (a) Distribution of layers of accepted streams at the viewers, and (b) distribution of the number of accepted streams at the viewers.

To evaluate the quantity of the accepted streams at each viewer, we plot the CDF of the viewers in terms of the number of accepted streams with similar setup. The result is shown in Figure 6.11(b). In this experiment, we also assign the outbound bandwidth uniformly between 0 to 12Mbps to the viewers and the number of viewers are varied from 10 to 1000. As the figure shows, most of the viewers (above 70%) receive all streams requested in a view with CDN capacity of 6000Mbps. Only 15% viewers do not receive any requested streams due to the bandwidth limitation.

6.5.4 Impact of Delay Variation and Buffer Size

The buffer size at the non-immersive viewers has impact on the violation of stream dependency. Higher the buffer size, lower the number of violation in the dependency preservation. By varrying the buffer size, we compare the performance of 4D TeleCast stream subscription algorithm with a "vanila" implementation, where only buffers and no caches are used at the viewers to preserve dependency. We compute *dependency violation* as the ratio of number of streams violated dependency to the number of streams received at the viewers. We run the experiments for $10,000$ viewers for 5 immersive participants and 4 video streams per participant. We use 5sec cache size in the stream subscription process. The outbound bandwidth is distributed uniformly between 10 to 50Mbps among the non-immersive users. Figure 6.12 shows the performance of

stream subscription algorithm over the vanila approach. For 500ms of buffer size, 4D TeleCast makes 55% less violation (i.e., no dependency violation) in the dependency preservation. The dependency violation is 36% less for 1sec of buffer size.

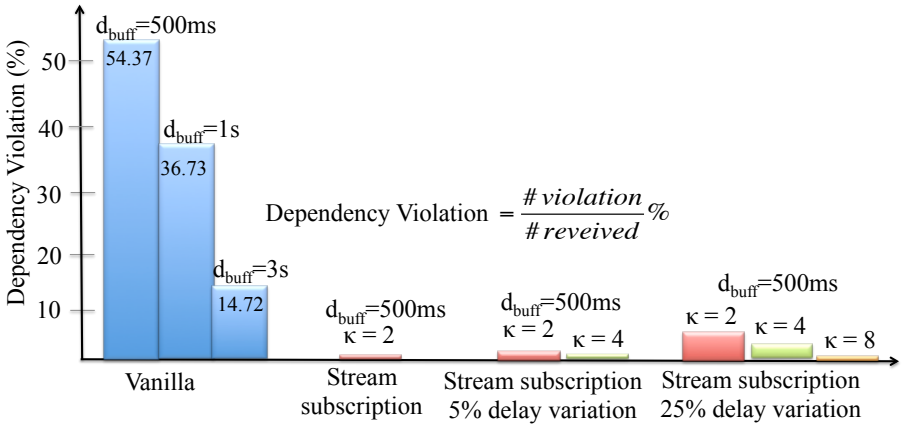


Figure 6.12: Impact of buffer size and delay variation in 4D TeleCast stream subscription.

Another factor that impacts the dependency at the viewers is the presence of jitter. A large variation in delay during the session run-time may make the layer bound among the dependent streams higher than κ. As Figure 6.12 shows, 5% and 25% variatons in the delay introduce about 3% and 8% dependency violation in 4D TeleCast for κ=2. To overcome this, the placement of the viewers within a delay layer should be precise so that the variation does not change the respective layer boundary. To make precise layering, we need to reduce the layer width, which leads to the higher value of κ. In our experimentation, κ=4 for 5% delay variation and κ=6 for 25% delay variation protect the dependency violation.

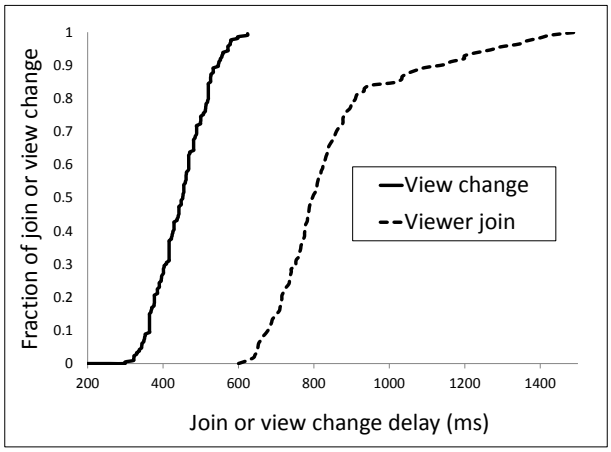


Figure 6.13: Overhead of 4D TeleCast for viewer join and view change.

6.5.5 4D TeleCast Overhead

The overhead of 4D TelCast comes from the overlay construction and the stream subscription processes. The join of a viewer goes through three steps: bandwidth allocation, overlay construction and stream subscription. Hence, the joining delay includes the network delay and the processing delay at each of these steps. Figure 6.11(c) shows the joining delay in terms of the CDF of the number of viewers. The value varies up to 1.5sec. The value shown in the Figure 6.11(c) does not include the buffering delay.

4D TeleCast improves the view change latency by serving the new streams due to the view changes directly from the CDN, while in the background, the new requests are added into the overlay using normal join steps. Therefore, the view change is satisfied quickly within 500ms. Some cases the view changes occur within 300ms.

6.6 Conclusion

In this chapter, we present 4D TeleCast, a novel multi-stream content dissemination framework. It supports a large number of concurrent views as well as viewers while preserving the unique nature of 3DTI multi-stream dependencies by ensuring maximum effective resource utilization and view change capabilities. Our research results are significant for next-generation multi-stream and multi-view 3D content distribution to a large number of concurrent users.

7 Conclusion and Future Work

Over the last few years, with the increased high-speed wired and wireless network availability, video traffic has quickly become the dominant fraction of Internet data traffic. Though a significant portion is coming from video-on-demand applications (such as Netflix and YouTube), traffic from interactive 2D video conferencing systems (such as Skype, Google Hangout, and Cisco tele-presence) is also very prominent. Moreover, current research efforts from both academia [105] [106] [107] [17] and industry [6] [108] [109] are adopting multi-camera, multi-site, and multi-modal 3D tele-immersive (3DTI) platforms for various online applications such as video conferencing, multi-player gaming, remote learning, collaborative dancing and tele-therapy. Projections show that by 2014, video traffic will constitute more than 90% of the total traffic on Internet [110].

7.1 Thesis Achievement

Unobtrusive and Fast TI Query Plane: We develop a monitoring control plane considering the hierarchical structure of 3DTI environments and the real-time requirement of application media traffic. It serves complex composition of multi-attribute based performance queries using a single monitoring overlay. Moreover, only a single query is disseminated into the monitoring overlay for any complex composition of multi-attribute ranges, which ensures fast and unobtrusive query lookup during the session run-time. Finally, our monitoring solution decides on the best way to store monitoring metadata by formalizing a relationship between the query rate and the metadata update rate so that stale results can be skipped.

User Expectation-aware Control Plane: Based on our earlier study of TI activity among the immersive users, we have observed that user expectations and hence quality of service or QoS requirement varies across activities. In this dissertation, we formalize the dependency of QoS specification and propose a novel evolutionary-based QoS optimization algorithm for immersive session. Our subjective evaluation shows that over 90% of the cases, users prefer the optimized session over QoS allocation without considering user expectations.

Large-scale Non-immersive Dissemination: Though the existence of immersive users are trivial in 3DTI applications, we explore another user dimension in this dissertation research, the non-immersive users, who require on-demand 3D streaming from immersive users and can be of large number. We show that without careful consideration in

streaming, we can waste network bandwidth for the delivery of streams which eventually are dropped at the viewers' display to preserve multi-stream dependency. We show that by designing a distributed stream subscription algorithm, it is possible to improve dependency violation over 55% depending on the buffer size at the users.

Feasible Application-Network Synergy: Using our session management framework, we have enabled a synergy between session and network layers so that application can drive network layer to support different application features in real-time.Using the combination of IP multicast at the edges and application multicast over WAN, we show control over some parts (last-mile) of the end-to-end Internet data paths. Our SDN-based data plane configuration enhances 3DTI interactivity, and reduces last-mile resource consumption in the network.

Seamless View Adaptation: When we use SDN-based data plane configuration over WAN, it becomes hard to maintain seamless real-time streaming experience at the victim sites as we need to update distributed switch states to reflect the new multi-stream distribution topology required by a view change. Since the view change can be very frequent, victim sites may experience frequent motion jerkiness and frame drop. To overcome it, in our cross-layer session-network control framework, we develop a sequencing algorithm for route updates in the distributed software defined switches, which ensures that view change events are atomic and do not create temporal disconnectivity or sudden burst in the network. In short, our solution provides a seamless viewing experience for the victim users.

7.2 Future Work

The necessity for the next generation of communication will never end. A fully automatic QoS management system for 3DTI environments is still an open issue. Here we discuss several of the most interesting and challenging research directions that we still have to solve in this domain.

Constructing Standard Database for Mapping TI Activity to QoS Profile: Our focus in the immersive domain is to construct a model that can reflect required QoS specifications into running session configuration. To prove our evolutionary solution, we have constructed QoS profiles for a limited set of 3DTI activities. Future research can be benefitted by exploring QoS profiles for different other TI activities and constructing a standard database that maps a TI activity to its corresponding QoS profile.

Predicting User Behavior to Improve Session Adaptation: To solve multi-stream prioritized QoS allocation, we use genetic optimization (GA) algorithm, which is a genre of evolutionary algorithm. Since GA is an iterative solution, sometimes it becomes slow to converge to find an optimal solution. To overcome the latency issue in 3DTI session adaptation, we introduce a two-step session adaptation process, which allows GA-based adaptation (second step) to run in parallel when a local adaptation (first step) is performed instantaneously. However, we can use several other extensions to improve the performance of session adaptation process using statistical learning meth-

ods. If we can predict the view change behavior of the participants, we can mask the optimization latency of GA for session adaptation by pre-computing the new multi-stream dissemination graph.

Extending Support for Application Features in Network Layer: Since SDN makes the network layer accessible to the application layer, it gives flexibility to the application developers to perform dynamic QoS control during application run-time. We show that using SDN we can easily construct a synergy to control network layer components to support different features of multimedia applications. We have investigated some of the QoS properties such as multiple forwarding and consistent network updates in our cross-layer session management framework. However, heterogenous rate assignment across multiple forwarding path from the switch, maintaining application frame at the network layer, detection early congestion and bandwidth provisioning are some of the possible QoS dimensions that we need to consider in future to further improve performance of interactive applications using SDN support.

Reducing Latency in Route Switching: Currently to perform consistent updates among the network switches distributed over WAN, a two-step communication is required. Further investigation needs to be done to reduce this route switching latency without violating the routing consistency requirement.

Allowing 3D Broadcasting: Finally, we have designed an early stage of 3D video broadcasting paradigm. Possible application domain may expand towards 3D live broadcasting of sports, where viewers can chose their virtual seats in the virtual environment based on the view orientation. Our presented framework therefore provides a conceptual basis for standardizing session management for multi-stream and multi-view large-scale 3D dissemination which does not require real-time response.

References

[1] pptv, "PPLive," http://www.pptv.com/.

[2] X. Zhang, J. Liu, B. Li, and Y. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," in *Proc. of International Conference on Computer Communications (ICCCN)*, 2005.

[3] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid CDN-P2P system for live video streaming: Experiences with LiveSky," in *Proc. of ACM Multimedia (MM)*, 2009.

[4] Z. Yang, W. Wu, K. Nahrstedt, G. Kurillo, and R. Bajcsy, "Enabling multi-party 3D tele-immersive environments with viewcast," in *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 2010.

[5] Hp Halo, ", http://www.hphalo.org."

[6] "Cisco 7200 series router architecture," http://www.cisco.com/en/US/products/hw/routers.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," in *RFC 3261*, 2002.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications, (RFC 3550)," 2003.

[9] International Telecommunication Union, "Packet based Multimedia Communication Systems Recommendation," h.323.

[10] D. Kahneman, "Thinking, fast and slow," 2011.

[11] B. Yu, "MyView: customizable automatic visual space management for multi-stream environment," *PhD Thesis*, 2006.

[12] Z. Zhang, D. Chu, J. Qiu, and T. Moscibroda, "Demo: Sword fight with smartphones," in *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011.

[13] R. Sheppard, M. Kamali, R. Rivas, M. Tamai, Z. Yang, W. Wu, and K. Nahrstedt, "Advancing interactive collaborative mediums through tele-immersive dance (TED): a symbiotic creativity and design environment for art and computer science," in *Proc. of ACM Multimedia (MM)*, 2008.

[14] G. Kurillo, M. Forte, and R. Bajcsy, "Tele-immersive 3D collaborative environment for cyberarchaeology," in *Proc. of IEEE CVPR workshop on Applications of Computer Vision in Archaeology*, 2010.

[15] G. Kurillo, T. Koritnik, T. Bajd, and R. Bajcsy, "Real-time 3D avatars for tele-rehabilitation in virtual reality," *Studies in health technology and informatics*, 2011.

[16] A. Maimone and H. Fuchs, "Real-time volumetric 3D capture of room-sized scenes for telepresence," in *Proc. of 3DTV-CON*, 2012.

[17] W. Wu, Z. Yang and I. Gupta and Klara Nahrstedt, "Towards multi-site collaboration in 3D tele-immersive environments," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2008.

[18] A. Arefin, Z. Huang, R. Rivas, S. Shi, P. Xia, K. Nahrstedt, W. Wu, G. Kurillo, and R. Bajcsy, "Classification and analysis of 3D tele-immersive activities," *IEEE Multimedia*, 2013.

[19] W. Wu, A. Arefin, Z. Huang, P. Agarwal, S. Shi, R. Rivas, and K. Nahrstedt, "I'm the Jedi! - A case study of user experience in 3D tele-immersive gaming," in *Proc. of International Symposium on Multimedia (ISM)*, 2010.

[20] A. Arefin, Z. Huang, R. Rivas, S. Shi, W. Wu, and K. Nahrstedt, "Tele-immersive gamings for everybody," in *Proc. of ACM Multimedia (MM)*, 2011.

[21] W. Wu, Z. Yang, and K. Nahrstedt, "Dynamic overlay multicast in 3d video collaborative systems," in *Proc. of Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2009.

[22] M. A. Arefin, Y. S. Udding, I. Gupta, and K. Nahrstedt, "Q-Tree: multi-attribute based range query solution for tele-immersive framework," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2009.

[23] A. Arefin, R. Rivas, and K. Nahrstedt, "Prioritized evolutionary optimization in open session management for 3D tele-immersion," in *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2013.

[24] Software-Defined Networking, http://bit.ly/LpV2lD, 2012.

[25] A. Arefin, R. Rivas, R. Tabassum, and K. Nahrstedt, "OpenSession: SDN-based cross-layer multi-stream management protocol for 3D teleimmersion," in *Proc. of International Conference on Network Protocols (ICNP)*, 2013.

[26] A. Arefin, Z. Huang, K. Nahrstedt, and P. Agarwal, "4D TeleCast: Towards large scale multi-site and multi-view dissemination of 3dti contents," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[28] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Renessei, "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead," in *Proc. of International workshop on Peer-To-Peer Systems (IPTPS)*, 2003.

[29] S. Deering, D. L. Estrin, D. Farinacci, and V. Jacobson, "The PIM architecture for wide-area multicast routing," *Transactions on Networking*, 1996.

[30] H. Eriksson, "Mbone: the multicast backbone," *ACM Communications*, 1994.

[31] M. Handley and V. Jacobson, "Session description protocol (RFC 2327)," 2006.

[32] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol," in *RFC 2326*, 1998.

[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[34] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of International Conference on Digital Signal Processing (ICDSP)*, 2001.

[35] M. Harren, J. Hellerstein, R. Huebsch, S. S. B. Loo, and I. Stoica, "Complex queries in dht-based peer-to-peer networks," in *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[36] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram, "'p-ring: An efficient and robust p2p range index structure," in *Proc. of SIGMOD*, 2007.

[37] R. Renesse and K. Binnanr, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transaction on Computer Systems (TOCS)*, 2001.

[38] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2004.

[39] PlanetLab 4hr traces, "http://www.eecs.harvard.edu/ syrah/nc/sim/pings.4hr.stamp.gz."

[40] CoMon: A Monitoring Infrastructure for PlanetLab, http://comon.cs.princeton.edu/, 2013.

[41] J. Albrecht, C. Tuttle, A. Snoeren, and A. Vahdat, "Planetlab application management using plush," in *Proc. of ACM SIGOPS Operating System Review (OSR)*, 2006.

[42] J. Liang, S. Ko, I. Gupta, and K. Nahrstedt, "Mon: On-demand overlays for distributed system management," in *Proc. of WORLDS*, 2005.

[43] S. Ko, S. Yalagandula, I. Gupta, V. Talwar, D. Milojicic, and S. Iyer, "Moara: Flexible and scalable group-based querying system," in *Proc. of ACM/IFIP/USENIX Middleware*, 2008.

[44] X. Chen, M. Chen, B. Li, and J. Li, "Celerity: a low-delay multi-party conferencing solution," in *Proc. of ACM Multimedia (MM)*, 2011.

[45] B. Wang and J. Hou, "Multicast routing and its QoS extension: problems, algorithms, and protocols," *IEEE Network*, 2000.

[46] F. Kuipers, P. Mieghem, T. Kokrmaz, and M. Krunz, "An overview of constraint-based path selection algorithms for QoS routing," *IEEE Communication Magazine*, 2002.

[47] M. Hosseini and N. Georganas, "3D videoconferencing application over an end system multicast protocol," in *ACM Multimedia (MM)*, 2003.

[48] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. of International Conference on Computer Communications (ICCCN)*, 2003.

[49] Z. Wang and J. Crowcroft, "Quality of service routing for supporting multimedia applications," *IEEE Journal of Selected Areas in Communications*, 1996.

[50] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proc. of Symposium on Operating Systems Principles (SOSP)*, 2003.

[51] Z. Huang, W. Wu, K. Nahrstedt, A. Arefin, and R. Rivas, "Tsync: a new synchronization framework for multi-site 3d tele-immersion," in *Proc. of ACM Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2010.

[52] J. A. Pouwelse, P. Garbacki, D. Epema, and H. J. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Proc. of International workshop on Peer-To-Peer Systems (IPTPS)*, 2005.

[53] L. Zhao, J.-G. Luo, M. Zhang, W.-J. Fu, J. Luo, Y.-F. Zhang, and S.-Q. Yang, "Gridmedia: A practical peer-to-peer based live video streaming system," in *Proc. of Workshop on Multimedia Signal Processing (MMSP)*, 2005.

[54] M. Zhang, L. Sun, X. Xi, and S. Yang, "igridmedia: Providing delay-guaranteed peer-to-peer live streaming service on internet," in *Proc. of Global Communication Conference (GLOBECOM)*, 2008.

[55] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peers," *Tech.Rep. 2002-21, Stanford University*, 2002.

[56] S. Jin and A. Bestavros, "Cache-and-relay streaming media delivery for asynchronous clients shudong," in *Proc. of International Workshop on Networked Group Communication (NGC)*, 2002.

[57] D. A. Tran, K. Hua, and T. Do, "Layered range multicast for video on demand," in *Proc. of International Conference on Computer Communications and Networks (ICCCN)*, 2002.

[58] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous media sharing in multimedia database systems," in *Proc. of International Conference on Database Systems for Advanced Applications(DASFAA)*, 1995.

[59] A. Dan and D. Sitaram, "A generalized interval caching policy for mixed interactive and long video workloads," in *Proc. of SPIE MultimediaComputing and Networking Conference (MMCN)*, 1996.

[60] P. L. Montessoro, "Efficient management and packets forwarding for multimedia flows," *Proc. of International Conference on Network and System Management*, 2012.

[61] E. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource Reservation ProtocolRFC 2205," 1997.

[62] R. Vogel, R. G. Herrtwich, W. Kalfa, H. Wittig, and L. C. Wolf, "Qos-based routing of multimedia streams in computer networks," *Selected Areas in Communications*, 1996.

[63] L. Toni, T. Maugey, and P. Frossard, "Multi-view video packet scheduling," arXiv:1212.4455.

[64] E. Rosen and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs) RFC 4364," 2006.

[65] K. Andreev, B. M. Maggs, A. Meyerson, and R. K. Sitaraman, "Designing overlay multicast networks for streaming," in *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2003.

[66] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," in *Proc. of SIGCOMM Computer Communication Review (CCR)*, 2008.

[67] M. Othman and K. Okamura, "Design and implementation of application based routing using OpenFlow," in *Proc. of International Conference on Future Internet Technologies (CFI)*, 2010.

[68] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: an open framework for OpenFlow switch evaluation," in *Proc. of Passive and Active Measurement (PAM)*, 2012.

[69] N. Handigol, S. Seetharaman, M. Flajslik, N. Mckeown, and S. Shenker, "Plug-n-server: Load balancing web traffic using OpenFlow," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2009.

[70] R. Wang, D. Batnariu, and J. Rexford, "Openflow based load balancing gone wild," in *Proc. of Hotice*, 2011.

[71] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of entire network (and its hosts)," in *Proc. of Hotnets*, 2012.

[72] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proc. of HotSDN*, 2012.

[73] H. Egilmez, B. Gorkemli, A. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *Proc. of International Conference on Image Processing (ICIP)*, 2011.

[74] M. Shand, S. Bryant, S. Previdi, C. Filsls, P. Francois, and O. Bonaventure, "Loop-free convergence using ofib," in *draft-ietf-rtgwg-ordered-b-06*, 2012.

[75] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2012.

[76] A. Corradi, L. Leonardi, and F. Zambonelli, "'diffusive load-balancing policies for dynamic applications," in *Proc. of IEEE Concurrency*, 1999.

[77] "PlanetLab 24 hr traces," http://www.eecs.harvard.edu/ syrah/nc/sim/ pings.4hr.stamp.gz.

[78] D. Kumar, D. Kashyap, K. Mishra, and A. Mishra, "Routing path determination using QoS metrics and priority based evolutionary optimization," in *Proc. of International Conference on High Performance Computing and Communications (HPCC)*, 2011.

[79] S. Upadhyaya and G. Dhingra, "Exploring issues for QoS based routing algorithm," *Journal of Computer Science and Engineering*, 2010.

[80] The Perceptual Evaluation of Speech Quality, "http://www.itu.int/rec/T-REC-P.862/en," 2001.

[81] M. Garey and D. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *Freeman, San Francisco*, 1979.

[82] F. Xiang, L. Junzhou, W. Jieyi, and G. Guanqun, "QoS routing based on genetic algorithm," *Computer Communications*, 1999.

[83] Z. Huang, A. Arefin, P. Agarwal, K. Nahrstedt, and W. Wu, "Towards the understanding of human perceptual quality in tele-immersive shared activity," in *Proc. of ACM Multimedia Systems (MMSys)*, 2012.

[84] F. Bauer and A. Varma, "Degree-constrained multicasting in point-to-point networks," in *International Conference on Computer Communications (ICCCN)*, 1995.

[85] R. Ravi, M. Marathe, S. Ravi, D. Rosenkrantz, and H. Hunt, "Approximation algorithms for degree-constrained minimum cost network-design problems," *Algorithmica*, 2001.

[86] K. Deb, "Multi-objective optimization using evolutionary algorithms," *John Wiley & Sons*, 2010.

[87] K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan, "A fast and elitist multi objective genetic algorithms:NSGA-II," *Transactions on Evolutionary Computation*, 2002.

[88] OpenFlow, www.openflow.org.

[89] Floodlight, http://floodlight.openflowhub.org/.

[90] IBM G8264, http://ibm.co/WbWfHi.

[91] M. Appelman, M. De Boer, and E. Van Der Pol, "Performance analysis of openflow hardware," http://bit.ly/11SnlGt.

[92] B. Hedlund, "On data center scale, OpenFlow, and SDN," http://bit.ly/gLJcIz.

[93] W. Beyda and R. Bitzinger, "System and method for reinterpreting tos bits," USPatent7106737, 2006.

[94] OpenFlow controller benchmark, http://bit.ly/upa5m7.

[95] OpenSession Seamless Route Update, http://www.youtube.com/watch?v=6XcFoJhkkL8.

[96] D. Xu, S. S. Kulkarni, C. Rosenberg, and H.-K. Chai, "A CDN-P2P hybrid architecture for cost-effective streaming media distribution," *Computer Networks*, 2004.

[97] B. Li, S. Xie, Y. Qu, Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new CoolStreaming: principles, measurements and performance implications," in *Proc. of International Conference on Computer Communications (ICCCN)*, 2008.

[98] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. of International Conference on Computer Communications (ICCCN)*, 2002.

[99] Akamai, "http://www.akamai.com/."

[100] "Amazon CloudFront," http://aws.amazon.com/cloudfront/.

[101] M. Afergan, J. Wein, and A. LaMeyer, "Experience with some principles for building an internet-scale reliable system," in *Proc. of Workshop on Real, Large Distributed Systems (WORLDS)*, 2005.

[102] Y. Chen, C. Chen, and C. L, "A measurement study of cache rejection in P2P live streaming system," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2008.

[103] X. Hei, C. Liang, and J. Liang, "Design and deployment of a hybrid CDN-P2P system for live video streaming," *IEEE Transactions on Multimedia*, 2007.

[104] J. Kuo, C. Shih, and Y. Chen, "A dynamic self-adjusted buffering mechanism for peer-to-peer real-time streaming," *Journal of Intelligent Systems and Application*, 2011.

[105] J. Cha, M. Eid, and A. Saddik, "Touchable 3d video system," *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 2009.

[106] F. Chen, J. Liu, and Y. Zhao, "Collaborative view synthesis for interactive multi-view video streaming," in *Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2012.

[107] D. Ott and K. Mayer-Patel, "Coordinated multi-streaming for 3D tele-immersion," in *ACM Multimedia (MM)*, 2004.

[108] H. H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender, "Understanding performance in Coliseum, an immersive videoconferencing system," *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 2005.

[109] Virtual Sword Fight, "http://www.wired.com/gadgetlab /2012/07/smartphone-sword-fighting/," 2012.

[110] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated Internet video control plane," in *Proc. of International conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2012.